UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

Desarrollo de componentes para la plataforma Keops.

Autor/es

MILLÁN SANTAMARÍA SACRISTÁN

Director/es

JESÚS MARÍA ARANSAY AZOFRA

Facultad

Facultad de Ciencia y Tecnología

Titulación

Grado en Ingeniería Informática

Departamento

MATEMÁTICAS Y COMPUTACIÓN

Curso académico

2019-20

publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

UNIVERSIDAD DE LA RIOJA

Facultad de Ciencia y Tecnología

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Desarrollo de componentes para la plataforma Keops.

Realizado por:

Millán Santamaría Sacristán

Tutelado por:

Jesús M.ª Aransay Azofra

Logroño, junio, 2020

Resumen

El presente Trabajo Fin de Grado tiene como objetivo realizar una API para la carga de modelos de inteligencia artificial para el análisis emocional. Estos modelos ya se habían comenzado a desarrollar antes de que yo llegara a la empresa, ya que esta es su tarea principal.

El resultado de este trabajo es un software que nos permitirá publicar de forma dinámica modelos de inteligencia artificial para segmentar el posible mercado para la venta de un producto o servicio.

También se generará la documentación de este proceso de publicación de nuevos servicios para permitir que usuarios no familiarizados con la API puedan publicar sus propios modelos de forma simple.

Finalmente, la API se publicará haciendo uso de las tecnologías Docker y Kubernetes.

Abstract

This Final Degree Project is aimed to carry out an API to load artificial intelligence models for emotional analysis. These models had already been developed by the company when I arrived there, since this is its main task.

The outcome of the project is software which will allow us to publish artificial intelligence models dynamically, in order to segment the potencial market for sale of a product or service.

All documentation of this new service publishing process will also be generated to enable users, who are unfamiliar with the API, to publish easily their own models.

Finally, the API will be published using Docker and Kubernetes technologies.

Contenido

Introducción

En las últimas décadas el avance de las tecnologías para la producción de componentes hardware ha permitido que la potencia computacional de los dispositivos haya aumentado de forma exponencial a lo largo de los años. La ley de Moore enunciada en 1975 por el cofundador de la empresa Intel, Gordon Moore la cual lleva su apellido dice "*el número de transistores por unidad de superficie en circuitos integrado se duplicará cada año*".

Este aumento de la capacidad de computación ha permitido un auge en el campo de la inteligencia artificial que a lo largo de los años se había visto siempre frenado por la falta de potencia computacional. Además de esto la información disponible en el mundo aumenta de manera exponencial, por el momento ya hay más bits almacenados de información que estrellas se cree que existen en el universo y para el año 2023 se estima que se llegará a superar el número de Avogadro (6.022 x 10^23). Esto sumado al aumento de la potencia de computación nos permite disponer de los recursos para entrenar los modelos de inteligencia artificial, así como disponer de una gran cantidad de información para entrenarlos.

Esto hizo que la empresa viera la oportunidad de extraer conocimiento acerca de los gustos de los individuos haciendo uso de las redes sociales, las cuales se han convertido en una parte inevitable de nuestras vidas. De esta información extraída se generarán "leads". Un lead es una persona o compañía que ha demostrado interés en la oferta de la marca, un potencial interesado en ciertos productos o servicios. Mi empresa se encargará de segmentar estos leads para vender esta información a empresas para que las mismas puedan dirigir campañas de marketing a aquel grupo de personas que debido a sus cualidades y características tiene un alto potencial, o existe una alta probabilidad de que pueda llegar a ser en el futuro un consumidor del producto o servicio de dicha empresa.

Actores

Autoría

Este documento, así como el Trabajo de Fin de Grado han sido realizados por Millán Santamaría Sacristán, estudiante de Grado en Ingeniería Informática en la Universidad de La Rioja.

Tutores

Este trabajo ha sido tutorizado por Jesús María Aransay Azofra del Departamento de Matemáticas y Computación de la Universidad de La Rioja.

Clientes

"The Demanda Valley", mi actual empresa representada por mi tutor de prácticas y de Trabajo de Fin de Grado, Jesús Navajas, como cliente.

Planificación del proyecto

Requisitos previos

Para el desarrollo de este trabajo necesitamos el desarrollo de una API la cual sea capaz de publicar modelos emocionales, previamente desarrollados por otro compañero de la empresa, a modo de servicio. La API también deberá permitir la publicación de modelos de clusterización como un servicio de la API, que contará con un modelo que deberemos de desarrollar para clusterizar las salidas del modelo emocional. Además, contará con otros servicios publicados como son uno para la obtención de la metainformación de los modelos de clusterización y otro para la obtención de diferentes métricas de los métodos publicados por los servicios de la API. También la API dispondrá de una aplicación de loggin propia.

Luego se mejorará la API permitiendo la carga dinámica de los modelos de clusterización eligiendo cuales publicar mediante un fichero de configuración desde el cual se configurarán todos los parámetros necesarios.

Finalmente nos gustaría hacerla pública mediante las tecnologías Docker y Kubernetes, para facilitar su publicación y escalado.

Metodología

Para el desarrollo de este tipo proyecto se ha decidido usar una metodología de desarrollo basada en iteraciones. He decidido basarme en una metodología de desarrollo en iteraciones ya que en cada iteración iremos añadiendo nuevas funcionalidades o mejoras a la API que comenzamos a desarrollar en la iteración 1. Cada una de estas iteraciones estará formada por las mismas fases: análisis de requisitos, diseño, implantación… La definición de requisitos al comienzo del proyecto fue muy superficial, es por eso que son necesarias estas fases en cada una de las iteraciones para completar la captura de requisitos más en detalle, y posteriormente se diseñará e implementará. Por lo tanto, se ha decidido seguir una metodología de trabajo en iteraciones. En la Imagen 1 se puede ver lo que sería cada iteración y sus fases. Pero para la asignación de tareas hemos empleado una metodología ágil. El proyecto se dividirá en tres iteraciones, las cuales se pueden ver en la tabla 1:

	Título
Iteración 1	Servicio web para modelos emocionales.
Iteración 2	Servicio web para modelos de Clusterización.
Iteración 3	Carga dinámica.

Tabla 1. Entregables Software

- **Iteración 1 - Servicio web para modelos emocionales:**

  En esta iteración se desarrollará una API para publicar diferentes servicios, en la que publicaremos un modelo emocional.

- **Iteración 2 - Servicio web para modelos de Clusterización:**

  En esta segunda iteración se desarrollará un servicio para publicar modelos de clusterización, en el que habrá publicado un modelo que desarrollaremos, así como un servicio de métricas. También se desarrollará un log personalizado para la API.

- **Iteración 3 - Carga dinámica:**

  En esta tercera iteración se desarrollará una carga en tiempo de ejecución de los modelos, se generará la imagen Docker para la API y se añadirá un servicio de metainformación.

Debido a todo esto, el diagrama de Gantt que establecemos al principio será susceptible de sufrir modificaciones ya que se desconoce el alcance real del proyecto.

Para tratar de seguir esta metodología hemos empleado el software para la administración de las tareas de un proyecto, llamado Jira, el cual forma parte del ecosistema de aplicaciones de la empresa Atlassian. Este software nos ha ayudado a realizar de forma más precisa el seguimiento y control del desarrollo del Trabajo de Fin de Grado.

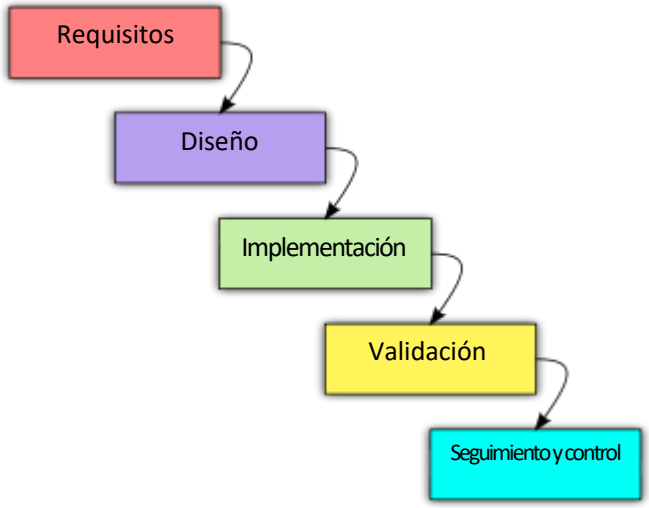

Imagen 1. Diagrama de la estructura en cascada de cada iteración

Metodología de control de versiones

En este punto explicaré qué servicios he utilizado para realizar un control de versiones. En primera instancia he creado un repositorio de "GitHub" para el tener un control de versiones del código. Este repositorio a su vez se usará internamente en la empresa para que cualquier futuro desarrollador tenga el código de la API que vamos a desarrollar. En cuanto a la documentación se ha creado una wiki para la empresa en la herramienta de Atlassian llamada Confluence. En ella se desarrollarán los diferentes manuales de uso y despliegue de la API.

Calendario y horario de trabajo

El presente Trabajo de Fin de Grado se comenzó durante el periodo de realización del TFG en la empresa el 17 de febrero de 2020. En la Tabla 2 se puede ver el calendario y el horario de trabajo realizado.

La otra limitación que nos impusimos fue defender el TFG en la convocatoria de junio, cuyas fechas de depósito son del 22 al 24 de junio.

Esquema de descomposición de tareas (EDT)

Puesto que en el momento de iniciar el desarrollo del proyecto no se conocía el alcance global del proyecto ni todo lo que se iba a desarrollar, es por este motivo que la EDT ha sufrido modificaciones a lo largo del proyecto.

El Diagrama 1 muestra la estructura de descomposición del trabajo de este proyecto. Está dividido en las diferentes iteraciones. Cada iteración es individual por lo que cada una constituye un proyecto por sí sola. Además, se ha reservado un bloque para el seguimiento y control del proyecto global.

Febrero						
Do.	Lu.	Ma.	Mi.	Ju.	Vi.	Sá.
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

Marzo						
Do.	Lu.	Ma.	Mi.	Ju.	Vi.	Sá.
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Abril						
Do.	Lu.	Ma.	Mi.	Ju.	Vi.	Sá.
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

Mayo						
Do.	Lu.	Ma.	Mi.	Ju.	Vi.	Sá.
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

	Lu.	Ma.	Me.	Ju.	Vi.
9:00 – 10:00					
10:00 – 11:00					
11:00 – 12:00					
12:00 – 13:00					
13:00 – 14:00					

Tabla 2. Calendario y horario de trabajo

Diagrama 1. Descomposición de Tareas (EDT)

En la siguiente tabla (Tabla 3) se muestra una descripción de los códigos presentes en la EDT y su correspondencia con los diferentes apartados de la memoria del Trabajo de Final de Grado.

Diagrama de Gantt

La siguiente tabla (Diagrama 2) muestra el diagrama de Gantt del proyecto. En él encontramos la estimación de semanas para completar cada una de las tareas y su distribución en el tiempo.

Como ya se ha comentado en el apartado "Esquema de descomposición de tareas", por la falta de información completa acerca del alcance del proyecto a su inicio, el alcance se irá definiendo en cada iteración. Es por este motivo que este diagrama de Gantt es meramente orientativo.

Código	Descripción
1.1	Definición de la iteración 1(Apartado 1.1).
1.2	Alcance de la iteración 1(Apartado 1.2)
1.3	Captura de requisitos de la iteración 1(Apartado 1.3)
1.4	Diseño de la iteración 1(Apartado 1.4)
1.5	Tecnologías de la iteración 1(Apartado 1.5)
1.6	Implementación de la iteración 1(Apartado 1.6)
1.7	Seguimiento y control de la iteración 1(Apartado 1.7)
1.8	Problemas de la iteración 1(Apartado 1.8)
2.1	Definición de la iteración 2(Apartado 2.1).
2.2	Alcance de la iteración 2(Apartado 2.2)
2.3	Captura de requisitos de la iteración 2(Apartado 2.3)
2.4	Diseño de la iteración 2(Apartado 2.4)
2.5	Tecnologías de la iteración 2(Apartado 2.5)
2.6	Implementación de la iteración 2(Apartado 2.6)
2.7	Seguimiento y control de la iteración 2(Apartado 2.7)
2.8	Problemas de la iteración 2 (Apartado 2.8)
3.1	Definición de la iteración 3(Apartado 3.1).
3.2	Alcance de la iteración 3(Apartado 3.2)
3.3	Captura de requisitos de la iteración 3(Apartado 3.3)
3.4	Diseño de la iteración 3(Apartado 3.4)
3.5	Tecnologías de la iteración 3(Apartado 3.5)
3.6	Implementación de la iteración 3(Apartado 3.6)
3.7	Seguimiento y control de la iteración 3(Apartado 3.7)
3.8	Problemas de la iteración 3(Apartado 3.8)

Tabla 3. Descripción de la descomposición de la EDT

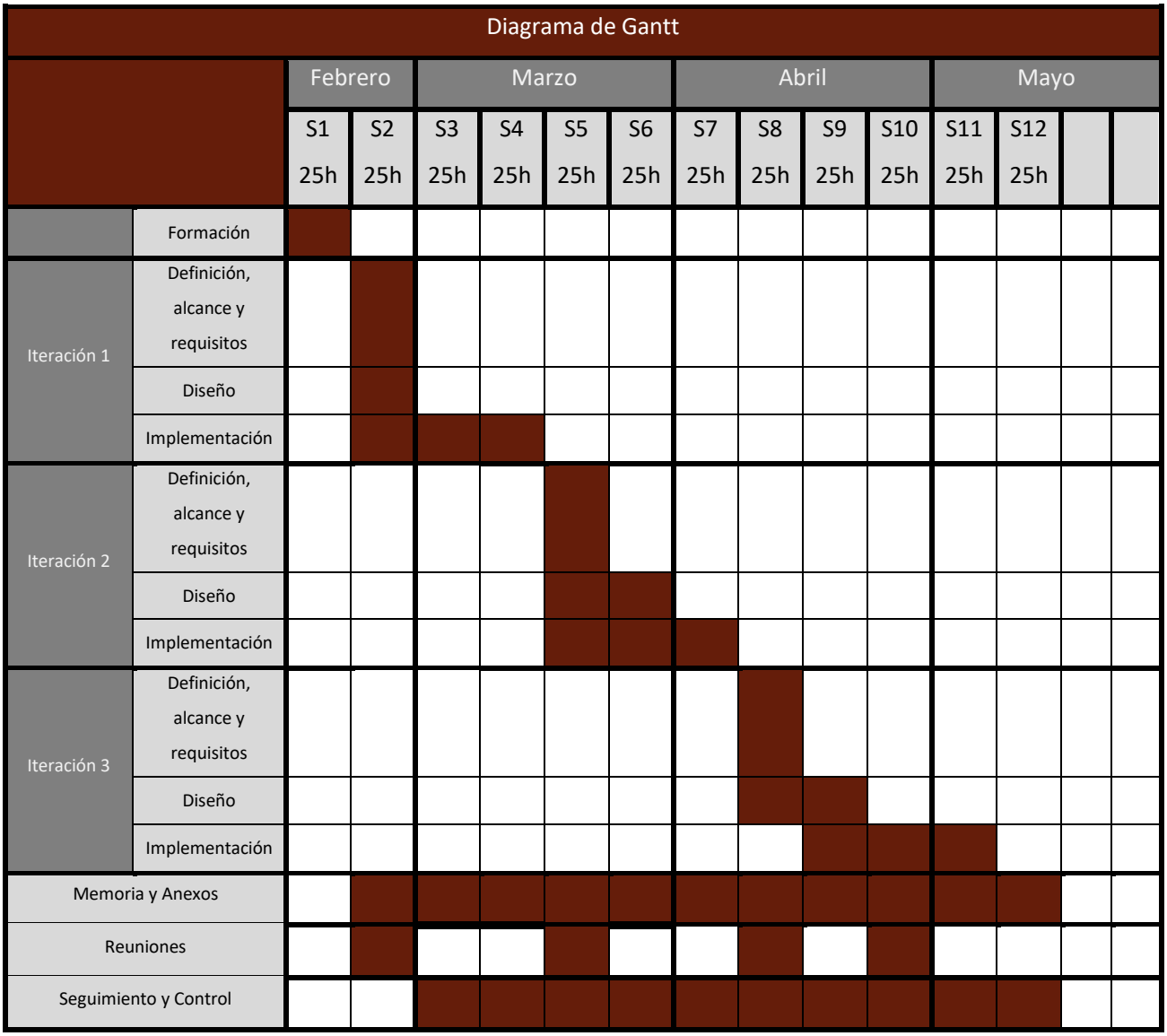

Diagrama 2. Diagrama de Gantt

Entregables

Como resultado del desarrollo de este trabajo se generarán los siguientes entregables de documentos (Tabla 4), con el código **DC(Documento)**.

Documentos:

Código	Descripción	Entrega
DC - 01	Memoria del Trabajo de Fin de Grado.	Universidad
DC - 02	Manual de uso de las diferentes componentes.	Empresa
DC - 03	Actas de reuniones.	Universidad

Tabla 4. Entregables Documentos

- **DC – 01 – Memoria del trabajo de Fin de Grado:**

  El presente documento, resultado de la realización del proyecto.

- **DC – 02 – Manual de uso de lo desarrollado en las diferentes iteraciones:**

  Documento el cual incluye la explicación de cómo utilizar lo desarrollado en las diferentes iteraciones. Se entregará a la empresa de modo que este pase a formar parte de la documentación interna de la empresa.

- **DC – 03 – Actas de reuniones:**

  Incluido como anexo en el presente documento. Condensarán lo tratado en las diferentes reuniones con el tutor de la empresa.

Plan de contingencia (Riesgos)

Durante la realización de este proyecto nos vamos a poder encontrar con una serie de riesgos durante su desarrollo, por lo que vamos a tratar de identificarlos antes de que estos sucedan (Tabla 5). En dicha tabla se muestra la fuente del riesgo, una breve descripción del mismo y su impacto en el correcto desarrollo del proyecto. En la tabla de identificación de soluciones (Tabla 6) se muestra la fuente y la descripción del riesgo y su plan de contingencia para minimizar su impacto.

Riesgos sobrevenidos

Durante el desarrollo del Trabajo de Fin de Grado surgieron una serie de problemas los cuales se identificaron y se trató de solucionar con el menor impacto para el normal desarrollo del TFG.

- **Retraso en el comienzo del TFG**
  Este riesgo ya se había identificado con antelación a su aparición. El comienzo del desarrollo se demoró dos semanas por parte de la empresa. Para tratar de minimizar su impacto se siguió el plan preestablecido, desarrollar la memoria del TFG. En este tiempo se pudo establecer una estructura base sobre la que poder comenzar a documentar.
- **Pandemia global**
  Este riesgo no se había identificado, surgió a mediados del desarrollo del Trabajo de Fin de Grado. En diciembre de 2019 en la ciudad china de Wuhan se detectó una enfermedad conocida como COVID-19 (o Coronavirus) la cual en pocos meses se ha convertido en una pandemia mundial, paralizando por completo numerosos países. Este riesgo no se contemplaba en ninguna situación, la situación actual de trabajo ha cambiado por completo. La solución hallada para tratar de minimizar su impacto en el ámbito laboral y en este TFG en concreto es el teletrabajo. Esta solución supuso trasladar mi puesto de trabajo a casa, trasladar el portátil de la oficina a mi casa y la comunicación con los tutores, tanto de la Universidad como de la empresa se siguió realizando por medios online como correos electrónicos y video llamadas.

Identificación de riesgos		
Fuente	**Descripción**	**Impacto**
Metodología	Cambio de los requisitos durante el desarrollo.	Medio
Empresa	Retraso en el comienzo del TFG.	Medio
Planificación	Superar las horas permitidas de TFG.	Medio

Tabla 5. Tabla de riesgos

Identificación de soluciones		
Fuente	**Descripción**	**Contingencia**
Metodología	Cambio de los requisitos durante el desarrollo	Se reevaluará la viabilidad del desarrollo del proyecto con los nuevos cambios
Externo	Retraso en el comienzo del TFG	Trabajar en la realización de la memoria de prácticas
Planificación	Superar las horas permitidas de TFG	Dejar sin realizar algún requisito el cuál no sea indispensable

Tabla 6. Tabla de solución de riesgos

1. Iteración 1: Servicio web para modelos emocionales

1.1. Definición de la iteración

Esta es la iteración inicial, en la cual se creará una API a la cual durante las distintas iteraciones se irán añadiendo distintos servicios web. En esta primera iteración se reservará un tiempo al aprendizaje de las diferentes tecnologías que se van a emplear en su desarrollo, y se añadirá el primer servicio a la API. Dicho servicio lanzará un modelo emocional de procesamiento de texto el cual nos devolverá una serie de valores que nos permitirá extraer conocimiento a través del texto sobre la persona. Además de un análisis de los Big-5[1], los cuales son una serie de dimensiones básicas de la personalidad: extroversión, afabilidad, responsabilidad, estabilidad emocional y apertura a nuevas experiencias. Y con ello tratar de analizar el comportamiento y personalidad de los individuos.

Este modelo emocional estará formado por cinco modelos TensorFlow para la extracción de cada una de las dimensiones básicas de la teoría de las Big-5. Estos modelos son redes neuronales que recibirán un vector con las medias basadas en el texto vectorizado median el lexicón LIWC. Estos modelos TensorFlow deducen la personalidad de quien ha escrito el texto.

"Linguistic Inquiry and Word Count" (LIWC) es un conjunto de diccionarios desarrollados para el análisis cognitivo y emocional de los textos gracias a una serie de variables psicológicas y estructurales. Se emplea en el campo de la psiquiatría, analiza los textos palabra por palabra, en un conjunto de 70 variables lingüísticas (Imagen 2), que incluyen variables del lenguaje estándar (artículos, preposiciones, pronombres, etc.), procesos psicológicos (categorías de emociones positivas y negativas, variables cognitivas), palabras relacionadas con la relatividad espacio-temporal (controlar el tiempo de un argumento es una buena forma de controlar su dirección) y los tiempos verbales.

LIWC fue desarrollado por James W. Pennebaker, profesor de psicología en la Universidad de Austin (Texas) en el año 2001. La premisa principal es que según la elección de palabras que hacemos da información sobre quien somos, a quién hablamos y en qué contexto lo hacemos, además de información sobre nuestro género, edad, estatus y motivaciones.

TensorFlow es una de las plataformas más importantes de aprendizaje profundo de la actualidad, propiedad de Google. Es una biblioteca de software de código abierto para la computación numérica, empleada para construir y entrenar redes neuronales.

TensorFlow fue desarrollado por el equipo que desarrollaba Google Brain[2], siendo TensorFlow el sistema de aprendizaje automático de la segunda versión de Google Brain, liberado como software de código abierto el 9 de noviembre del 2015.

La primera mención publica de la teoría de las Big-5 data de 1933 por Louis Leon Thurstone en su alocución del presidente ante la American Psychological Association. Esta teoría afirma que mediante el estudio de las cinco dimensiones de la personalidad básicas (extroversión, afabilidad, responsabilidad, estabilidad emocional y apertura a nuevas experiencias) podemos conocer el comportamiento y la personalidad de los individuos.

---

[1] **Big-5:** Es la clasificación de rasgos de personalidad que analiza la composición de cinco dimensiones de personalidad en su sentido más amplio. (https://positivepsychology.com/big-five-personality-theory/)

[2] **Google Brain:** https://research.google/teams/brain/

Con dicho servicio se pretenden analizar los tweets en Twitter de diversos usuarios, con la teoría psicológica de los Big-5, con el fin de generar leads segmentados para lanzar campañas publicitarias dirigidas. Estas campañas serán lanzadas en diferentes idiomas: español, inglés y portugués. Pero por el momento solo están desarrollados los modelos para español.

LIWC 2007 Dimensions and Sample Words

DIMENSION	EXAMPLES	# WORDS
STANDARD LINGUISTIC DIMENSIONS		
Total function words		464
Total pronouns	I, them, itself	116
Personal pronouns	I, them, her	70
1st person singular	I, me, mine	12
1st person plural	we, our, us	12
2nd person	you, your, thou	20
3rd person singular	she, her, him	17
3rd person plural	they, their, they'd	10
Impersonal pronouns	it, its's those	46
Articles	a, an, the	3
Verbs	walk, went, see	383
Auxiliary verbs	Am, will,have	144
Past tense	walked, were, had	145
Present tense	Is, does, hear	169
Future tense	will, gonna	48
Adverbs	very, really, quickly	69
Prepositions	with, above	60
Conjonctions	but, whereas	28
Negations	no, never, not	57
Quantifiers	few, many, much	89
Numbers	one, thirty, million	34
Swear words	damn, fuck, piss	53
PERSONAL CONCERNS		
Work	work, class, boss	327
Achievement	try, goal, win	186
Leisure	house, TV, music	229
Home	house, kitchen, lawn	93
Money	audit, cash, owe	173
Religion	altar, church, mosque	159
Death	bury, coffin, kill	62
SPOKEN CATEGORIES		
Assent	agree, OK, yes	30
Nonfluencies	uh, rr*	8
Fillers	blah, you know, I mean	9

PSYCHOLOGICAL PROCESSES		
Social Processes	talk, us, friend	455
Friends	pal, buddy, coworker	37
Family	mom, brother, cousin	64
Humans	boy, woman, group	61
Affective Processes	happy, ugly, bitter	915
Positive Emotions	happy, pretty, good	405
Negative Emotions	hate, worthless, enemy	499
Anxiety	nervous, afraid, tense	91
Anger	hate, kill, pissed	184
Sadness	grief, cry, sad	101
Cognitive Processes	cause, know, ought	730
Insight	think, know, consider	195
Causation	because, effect, hence	108
Discrepancy	should, would, could	76
Tentative	maybe, perhaps, guess	155
Certainty	always, never	83
Inhibition	block, constrain	111
Inclusive	with, and, include	18
Exclusive	but, except, without	17
Perceptual Processes	see, touch, listen	273
Seeing	view, saw, look	72
Hearing	heard, listen, sound	51
Feeling	touch, hold, felt	75
Biological Processes	eat, blood, pain	567
Body	ache, heart, cough	180
Health	clinic, flu, pill	236
Sexuality	horny, love, incest	96
Ingestion	eat, swallow, taste	111
Relativity	area, bend, exit, stop	638
Motion	walk, move, go	168
Space	Down, in, thin	220
Time	hour, day, oclock	239

Imagen 2. Salida LIWC

1.2.Alcance

La planificación inicial no permitía que el alcance completo de proyecto se definiera antes de comenzar. Puesto que en cada iteración se desarrollan servicios diferentes no supuso un mayor problema. La planificación inicial de tiempos sí que tuvo que ser revisada en cada iteración para así poder ajustar los tiempos a lo que se pretendía desarrollar en cada iteración.

Los objetivos que conseguir con esta iteración son los siguientes:

- Como se define en los requisitos, el desarrollo de un servicio web para interactuar con el modelo emocional.
- Desarrollo de un cliente para interactuar desde línea de comandos con la API que vamos a desarrollar.
- Generar la documentación para el modelo.

1.3.Captura de requisitos

La captura de requisitos se realizó a través de reuniones con el tutor de la empresa. (Ver anexo, Actas de reuniones). Estos se han dividido en los siguientes bloques: Requisitos funcionales, Requisitos no funcionales, Requisitos transversales y Exclusiones. La tabla constará de dos columnas, una para el **código** del requisito con el que se referirán los requisitos dentro del presente documento y la **descripción** de este.

1.3.1. Requisitos funcionales

En la Tabla 7 de detallan los requisitos no funcionales de la iteración 1, que después se detallan más abajo. Estos requisitos tendrán la siguiente forma, **RFE – XX (Requisitos funcionales Emocionales).**

Requisitos funcionales del servicio	
Código	**Descripción**
RFE -01	Creación una clase de carga para los distintos modelos ya entrenados que formarán el modelo emocional.
RFE -02	Creación de una clase con los diferentes métodos que queramos que publique nuestro servicio para interactuar con los modelos Emocionales.
RFE -03	Creación de las clases necesarias para la estandarización de la salida en formato JSON.
RFE -04	Creación de un método para procesar grandes cantidades de texto que debe de recibir la información como texto plano en el body de la petición.
RFE -05	Creación de un método el cual acepte JSON en el body de la petición.
RFE -06	Los modelos Emocionales para los diferentes idiomas deben de estar cargados al iniciar el servicio.
RFE -07	El servicio debe de funcionar con el modelo Emocional para el idioma español.
RFE -08	Creación de un cliente para interactuar con el servicio desde línea de comandos.
RFE -09	Todos los métodos solo deben aceptar peticiones POST.

Tabla 7. Requisitos funcionales iteración 1

RFE -01 - Creación de las clases de carga para los distintos modelos ya entrenados que formarán el modelo emocional:

Se desarrollarán dos clases, una para la carga de los modelos ya entrenados y otra para la vectorización de las palabras de los textos según el léxicon LIWC.

La clase de carga servirá para cargar los modelos previamente creados, entrenados y serializados en el formato h5[3], almacena los parámetros aprendidos por la red neuronal (pesos y bias), así como de interfaz para interactuar con los modelos. Esta clase depende por completo de la segunda clase que crearemos, la cual servirá para preprocesar los textos antes de pasárselos a los modelos. Los métodos de esta clase solo serán llamados desde los métodos de la clase de carga para interactuar con los modelos. Estos métodos se encargarán de tokenizar las palabras de los textos según los diccionarios LIWC previamente cargados con la clase anterior.

[3] **h5:** es un archivo de datos guardado en el Formato de Datos Jerárquicos (HDF). Contiene matrices multidimensionales de datos científicos. Los archivos H5 se usan comúnmente en el sector aeroespacial, física, ingeniería, finanzas, investigación académica, genómica, astronomía, instrumentos electrónicos y campos médicos.

El resultado de llamar a los métodos de la clase de carga será los valores que nos devuelve el modelo Emocional en formato JSON.

RFE -02 - Creación de una clase con los diferentes métodos que queramos que publique nuestro servicio para interactuar con los modelos Emocionales:

Se desarrollará una clase con los diferentes métodos que queramos que tenga el servicio que publique los modelos Emocionales. Cada uno de estos métodos estará asociado a una URL con la cual podremos interactuar con estos métodos. A estos métodos se les pasará como parámetros los textos que se quieren analizar, así como el idioma del texto a procesar. Y nos devolverá la salida del modelo emocional, al cual le hemos pasado como entrada el texto vectorizado con LIWC y nos devolverá la salida del modelo en formato JSON.

RFE -03 - Creación de las clases necesarias para la estandarización de la salida en formato JSON:

Se desarrollará una clase para procesar las salidas en formato JSON de la invocación a los métodos del servicio. Se estandarizarán estas salidas de acuerdo a un estándar previamente definido por nosotros. Se devolverá las salidas estandarizadas en formato JSON como salida de los métodos.

RFE -04 – Creación de un método para procesar grandes cantidades de texto que debe de recibir la información como texto plano en el body de la petición:

Este método del servicio web será invocado mediante una petición Post a su correspondiente URL, <URL método>/<código idioma>. En dicha petición POST, el body de la petición contendrá el texto que deseemos analizar y como variable del path de la URL de la invocación al método, el código del idioma del texto que queremos analizar, dicho código de idioma seguirá el estándar ISO 639-1[4]. Este método devolverá un JSON con las predicciones devueltas por el modelo Emocional.

RFE -05 – Creación de un método el cual acepte JSON en el body de la petición:

Este método del servicio web será invocado mediante una petición Post a su correspondiente URL, <URL método>/. En dicha petición POST, el body de la petición contendrá el texto a analizar y el código de idioma, el cual sigue el estándar ISO 639-1, en formato JSON. Este método devolverá un JSON con las predicciones devueltas por el modelo Emocional.

RFE -06 - Los modelos Emocionales para los diferentes idiomas deben de estar cargados al iniciar el servicio:

La carga de estos modelos es muy lenta, por lo que no es viable hacerlos en tiempo de ejecución. Tienen que estar todos cargados al arrancar el servicio para que no afecte a la eficiencia.

RFE -07 - El servicio debe de funcionar con el modelo Emocional para el idioma español:

Se debe cargar el modelo Emocional para el idioma español(es).

RFE -08 - Creación de un cliente para interactuar con el servicio desde línea de comandos:

Creación de un cliente de consola para hacer llamadas a los diferentes métodos disponibles en el servicio. Al ejecutar este cliente en una consola nos aparecerá un menú en el cual podremos elegir el tipo de petición que deseemos realizar. Nos pedirá los datos de entrada que sean precisos y nos mostrará la salida por la consola.

[4] **Estándar ISO 639-1:** es la primera parte del código ISO 639. Consiste en 184 códigos de dos letras usados para identificar los principales idiomas del mundo. (https://es.wikipedia.org/wiki/ISO_639-1)

RFE -09 - Todos los métodos solo deben aceptar peticiones POST:

Puesto que se envía información en las peticiones a los métodos, estos deben de aceptar únicamente peticiones Post.

1.3.2. Requisitos no funcionales

En la Tabla 8 de detallan los requisitos no funcionales de la iteración 1. Estos requisitos tendrán la siguiente forma, **RNE – XX (Requisitos no funcionales Emocionales)**

Requisitos no funcionales del servicio	
Código	**Descripción**
RNE -01	Entrega de la iteración 1 el 13/03/2020.
RNE -02	Uso de Python para desarrollo del servicio Web.
RNE -03	Uso de Flask como servidor web.
RNE -04	Uso de Flasgger para documentar los métodos del servicio.
RNE -05	El servicio debe ser multihilo.

Tabla 8. Requisitos no funcionales iteración 1

RNE -01 - Entrega del servicio de LIWC el 13/03/2020:

Entrega de la documentación generada acerca del servicio, así como el propio servicio.

RNE -02 - Uso de Python para desarrollo del servicio Web:

Emplear como lenguaje de programación Python, ya que los modelos que se van a cargar están desarrollados con este lenguaje. Esto hace que para hacer más simple su uso empleemos el mismo lenguaje que el de los modelos.

RNE -03 - Uso de Flask como servidor web:

Emplear Flask[5] como framework de desarrollo para crear los servicios web.

RNE -04 - Uso de Flasgger para documentar los métodos del servicio:

Emplear Flasgger[6] para documentar los métodos del servicio en formato HTML, además de proporcionar una forma de interactuar en línea de forma gráfica con ellos.

RNE -05 - El servicio debe ser multihilo:

El servicio debe de ser multihilo, es decir que acepte peticiones concurrentes. Para que esto sea posible los modelos Emocionales deben de serlo, por lo que esto es un requisito indirecto también.

[5] **Flask:** es un framework minimalista escrito en Python que permite crear aplicaciones web rápidamente y con un mínimo número de líneas de código. (https://flask.palletsprojects.com/en/1.1.x/)
[6] **Flasgger:** Flasgger es una extensión de Flask para extraer la especificación OpenAPI de todas las vistas de Flask registradas en su API. (https://github.com/flasgger/flasgger)

1.3.3. Requisitos transversales

En la Tabla 9 se detallan los requisitos transversales de la iteración 1. Estos requisitos tendrán la siguiente forma, **RTE – XX (Requisitos transversales Emocionales)**

RTE -01 - Validación del servicio por el tutor de la empresa:

El servicio desarrollado tiene que ser revisado y aceptado por el tutor de la empresa.

RTE -02 - Validación de la documentación del servicio por el tutor de la empresa:

La documentación generada acerca del servicio tiene que ser revisada y validada por el tutor de la empresa.

Requisitos transversales del servicio	
Código	**Descripción**
RTE -01	Validación del servicio por el tutor de la empresa.
RTE -02	Validación de la documentación del servicio por el tutor de la empresa.

Tabla 9. Requisitos transversales iteración 1

1.3.4. Exclusiones

En la Tabla 10 de detallan los requisitos excluidos de la iteración 1. Estos requisitos tendrán la siguiente forma, **EE – XX (Exclusiones Emocionales)**

Exclusiones del servicio	
Código	**Descripción**
EE -01	El servicio no debe de estar implementado con el modelo Emocional para el idioma inglés.
EE -02	El servicio no debe de estar implementado con el modelo Emocional para el idioma portugués.
EE -03	No formará parte de este incremento el desarrollo del Dockerfile.

Tabla 10. Exclusiones iteración 1

EE -01 - El servicio no debe de estar implementado con el modelo Emocional para el idioma inglés:

Está excluido que funcione con el modelo Emocional para el idioma inglés puesto que no está desarrollado por el momento dicho modelo. Este modelo se desarrollará más adelante.

EE -02 - El servicio no debe de estar implementado con el modelo Emocional para el idioma portugués:

Está excluido que funcione con el modelo Emocional para el idioma portugués puesto que no está desarrollado por el momento dicho modelo. Este modelo se desarrollará más adelante.

EE -03 - No formará parte de este incremento el desarrollo del Dockerfile:

Por el momento no será necesario, se creará cuando se vaya a subir a producción la aplicación de servicios.

1.4. Diseño

En esta interacción vamos a comenzar a desarrollar el servicio web, el cual es una API Rest en Python. En dicho servicio web tendremos una serie de métodos los cuales serán invocados

desde la URL que cada uno de ellos tenga asociada. Todas las URLs asociadas a los métodos relacionados con los modelos LIWC colgarán de la URL base "/liwc". En la Imagen 3. se muestra la estructura de dichas URLs.

/: Esta URL solo recibe peticiones POST, la cual acepta JSON en el body de la petición. Este JSON debe tener dos claves de tipo string: idioma y texto. El idioma debe de ser el código ISO 639-1 del idioma. Devuelve un JSON con todo el conocimiento extraído del modelo Emocional, anexo, Ejemplo de salida del modelo Emocional. Este método corresponde con el requisito funcional con código RFL -05.

/<string:idioma>: Esta URL solo recibe peticiones POST, a la cual se le pasa el código ISO 639-1 del idioma como un parámetro en el path y el texto a analizar en el body de la petición. Devuelve un JSON con todo el conocimiento extraído del modelo Emocional, anexo, Ejemplo de salida del modelo Emocional. Este método corresponde con el requisito funcional con código RFE -04. Este método lo propuso el CTO de la empresa para así evitar la sobrecarga de enviar en JSON textos grandes, ya que el generarlo es rápido pero el análisis es lento, de esta manera se envía plain-text en el body.

1.5. Tecnologías

Para esta iteración se han empleado las siguientes tecnologías:

1. **Python**[7] es el lenguaje con el cual desarrollaremos el servicio, ya que así lo ha pedido la empresa. La versión empleada es 3.7. no siendo esta la última versión disponible (3.8) por un problema de compatibilidad con la librería TensorFlow empleada para el modelo.
2. **TensorFlow**[8] es una librería empleada en aprendizaje automático para la construcción de modelos predictivos. Esta librería la emplearemos para construir y entrenar redes neuronales para tratar de detectar y descifrar patrones mediante la vectorización del texto con LIWC. Empleamos la versión 2.0.0.
3. **Flask** es un servidor web disponible en Python, con él lanzamos nuestro servicio.
4. **Flasgger**, implementa Swagger para Flask. Swagger es un gestor de APIS que puede incluir como en nuestro caso la autodocumentación de las APIS ofrecidas, permitiendo desde el propio servicio consultar los servicios disponibles e invocarlos sin necesidad de cliente. En nuestro caso lo emplearemos para hacer más fácil usarlo y probarlo.
5. **Blueprint**[9] esta librería la usaremos para organizar la estructura interna del servicio, permitiéndonos crear agrupaciones de URL's según directorios. Con ello podemos establecer una URL base diferente para cada servicio web.

[7] **Python:** Python es un lenguaje de programación interpretado multiparadigma. (https://www.python.org/)
[8] **TensorFlow:** Es una biblioteca de código abierto para aprendizaje automático. (https://www.tensorflow.org/)
[9] **Blueprint:** Esta librería nos permite organizar nuestra aplicación en diferentes módulos. (https://flask.palletsprojects.com/en/1.1.x/blueprints/)

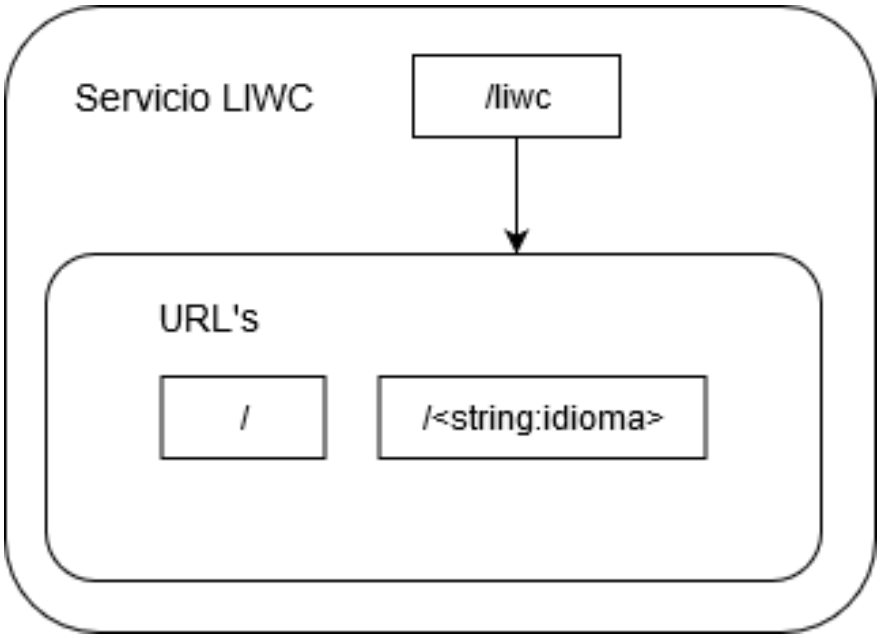

Imagen 3. Esquema del diseño de los servicios de la iteración 1

1.6.Implementación

El servicio web para modelos Emocionales se encuentra dentro de una aplicación web con la estructura que se puede ver en la Imagen 4. La aplicación web se lanza desde el fichero **entrypoint.py,** donde se encuentra el Main y se asocia Flasgger. En la carpeta **aplicacion** se encuentran dos ficheros: **errores.py**, ahí definimos y manejamos los posibles errores que puedan aparecer durante la ejecución (Por ejemplo, el error 501 Not Implemented). En el fichero **aplicacion/__init__.py** disponemos del método para crear la aplicación Flask, asociar las URLs a los métodos y el manejador de errores. También tenemos la carpeta **modelos** en la cual tendremos la carpeta **aplicación/modelos/modeloLIWC** donde tenemos todos los ficheros necesarios para ese servicio, los métodos asociados a las URLs así como los modelos ya entrenados, los cuales aceptan múltiples peticiones simultáneas permitiendo concurrencia. En el fichero **aplicación/modelos/modeloLIWC/__init__.py** es donde mapeamos las diferentes URLs con los métodos que se encuentran en **aplicación/modelos/modeloLIWC/runLIWC.py.** Dentro de la carpeta **estandarizacion** encontramos las clases **aplicación/modelos/modeloLIWC/estandarizacion/ estandarizacion.py** donde tenemos los métodos necesarios para estandarizar las salidas y **aplicación/modelos/modeloLIWC/estandarizacion/estandarizacion.py** corresponde con los diccionarios de como estandarizar dicha salida.

Cuando por ejemplo hacemos una petición a la URL “http://localhost:5000/liwc/es”, queremos llamar al método Emocional del idioma español como se indica en la URL. Dicha petición debe ser POST y en el cuerpo de la petición (body) debe contener el texto que se desea analizar. El

servidor recibirá esta petición y mediante la tecnología *blueprint* haremos llegar la petición al método asociado con esa URL. Dicho método obtendrá como parámetros el idioma (obtenido de la URL) y el texto (obtenido del cuerpo de la petición) a analizar. De la factoría de modelos ya cargados se obtendrá el que corresponda con el idioma, en este caso español, y con dicho modelo se hará una predicción. La predicción obtenida antes de poder trabajar con ella necesitamos estandarizarla, para ello de la misma factoría de modelos obtendremos el objeto con el que poder estandarizar la salida en formato JSON. Tras esta transformación devolveremos el resultado ya que Flask permite devolver JSON sin necesidad de serialización. Un ejemplo de la salida del modelo lo tenemos en el anexo, Ejemplo de salida del modelo emocional.

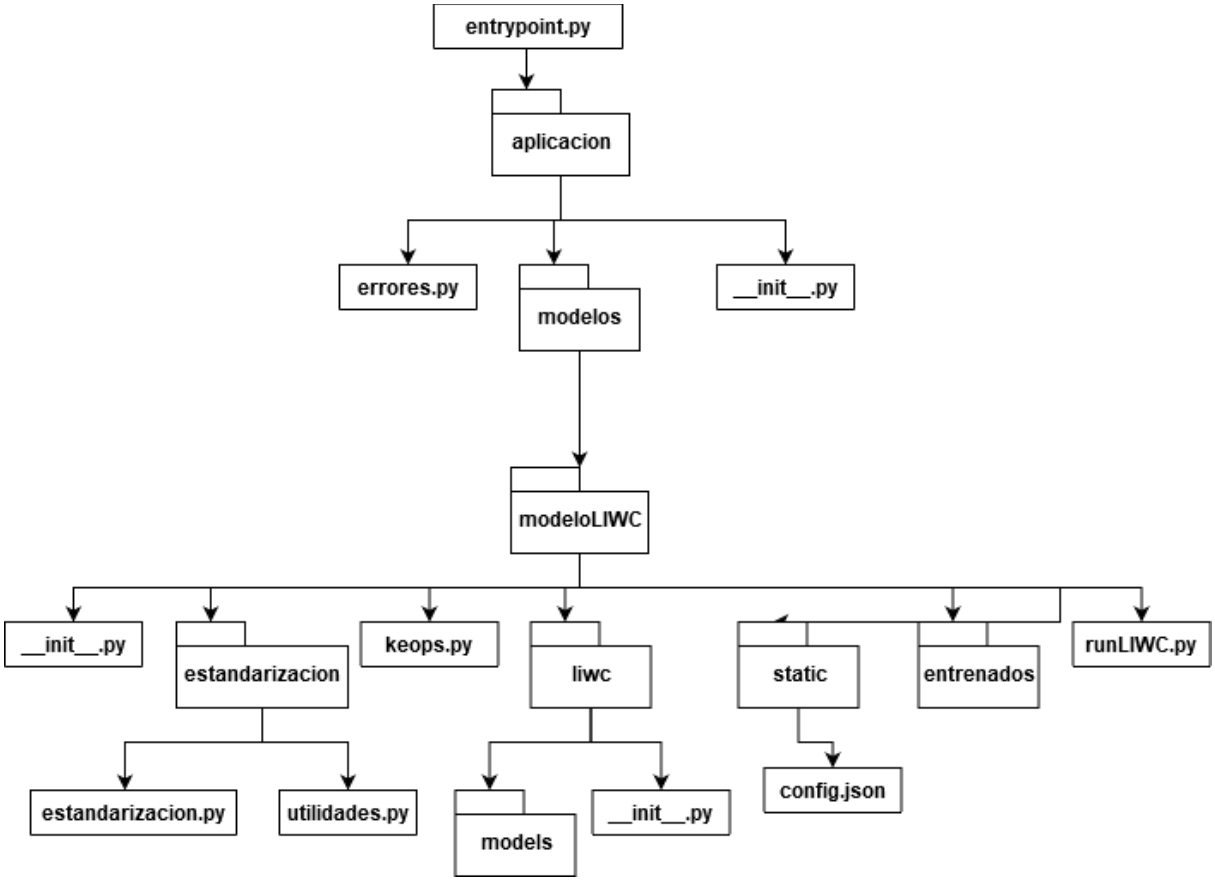

Imagen 4. Estructura de la aplicación web en la iteración 1

1.7.Seguimiento y control

En este apartado analizaremos el desarrollo de la iteración, así como sus desviaciones con respecto a la planificación inicial.

1.7.1. Seguimiento de cambios

Durante el desarrollo de esta iteración no surgió ningún cambio, por lo que el desarrollo de la iteración no se ha visto afectado por ellos.

1.7.2. Seguimiento del alcance

Al comenzar la iteración se establecieron una serie de requisitos los cuales se iban a tratar de desarrollar a lo largo de la iteración, es por ello que debemos analizar la situación del alcance al finalizar la iteración.

1.7.3. Requisitos alcanzados

Se han cumplido satisfactoriamente todos los requisitos especificados en el apartado "1.2. Alcance".

1.7.4. Entregables generados

Se ha desarrollado el siguiente entregable:

- **RNE-1 Documentación:** contiene las especificaciones del servicio web desarrollado en esta iteración, así como un manual de su uso e implementación del servicio. Esta documentación no está incluida como anexo en esta memoria puesto que es documentación interna de la empresa.

1.8. Problemas

Durante la realización de esta componente nos encontramos con los siguientes problemas, los cuales supusieron un retraso en la planificación:

Retraso con los modelos (13/02/2020 – 20/02/2020):

Este problema se debió a que mi compañero se retrasó con la entrega de los modelos ya entrenados una semana. Este tiempo lo empleé como tiempo de formación y para redactar la memoria del Trabajo de Fin de Grado. Este riesgo fue previsto en el apartado "Plan de contingencia".

Compatibilidad de la librería TensorFlow (21/02/2020 – 24/02/2020):

Este problema se debió a que la versión 2.0.0 de la librería con la que se habían creado los modelos Emocionales no era compatible con la versión 2.1.0 que estaba empleando. Hasta la versión 2.0.0 de TensorFlow las versiones superiores eran compatibles con las versiones anteriores, pero en esta versión se rompió esta compatibilidad. Durante este tiempo estuve tratando de instalar la versión 2.0.0 de la librería.

2. Iteración 2: Servicio web modelos de Clusterización

2.1. Definición de la iteración

En esta iteración desarrollaré un modelo de clusterización Gaussiano el cual es un modelo no supervisado, es decir que no emplean una salida esperada para entrenar. El modelo separará los datos buscando patrones o agrupaciones de los mismos. Emplearemos un modelo Gaussian Mixture (Mezcla Gaussiana) el cual es una función que se compone de varias funciones Gaussianas, cada una identificada por k ∈ {1, ..., K}, donde K es el número de clúster (grupo) del dataset (conjunto de datos). Cada una de estas funciones se compone de los siguientes parámetros (Imagen 5): la media para definir su centro, la covarianza para definir su ancho y una probabilidad de mezcla que define el tamaño de la función Gaussiana. Es un modelo de máxima verosimilitud, se busca la función con mayor probabilidad para cada punto[10]. Tras obtener el mejor modelo generaremos mediante un notebook [11] de Jupyter [12] un informe sobre los clústeres del modelo (Imagen 6) los cuales serán analizados y validados por el psicólogo de la empresa para comprobar si son segmentaciones lógicas de las personas y así poder poner en producción el modelo. Con este modelo se pretende segmentar a las personas para tratar de identificar los individuos a quien lanzar una campaña publicitaria para tratar de obtener los mejores resultados de ventas.

Este modelo se publicará en la API como un servicio. Se diseñará este servicio para permitir la carga de diferentes modelos de clusterización, pero por el momento solo publicaremos uno, el que desarrollaremos. El modelo de clusterización que publicaremos nos servirá para segmentar los diferentes tipos de personas que podemos encontrar. Para ello este modelo empleará la salida devuelta por el modelo Emocional desarrollado en la componente anterior (en realidad, solo empleará determinadas variables de la salida).

Además, se desarrollarán dos componentes para la aplicación. La primera es un servicio de log personalizado para el servicio web. Este además de mostrar por pantalla los diferentes avisos, los almacenará en ficheros de log permitiéndonos posteriormente recuperar información sobre las diferentes peticiones realizadas a nuestro servicio web. La otra componente será una serie de métricas de los diferentes métodos publicados como pueden ser la frecuencia de uso en diferentes tramos de tiempo (por ejemplo, número de peticiones por minuto a un método) o el número total de peticiones recibidas para así poder analizar el uso de la API. Estás métricas serán accesibles como un servicio de nuestra API.

[10] Más en profundidad queda explicado en el siguiente enlace https://towardsdatascience.com/gaussian-mixture-models-d13a5e915c8e .
[11] **Notebook:** es un documento JSON, que sigue un esquema versionado y que contiene una lista ordenada de celdas de entrada/salida que pueden contener código, texto (Markdown), matemáticas y gráficos.
[12] **Jupyter:** es un entorno informático interactivo web para crear notebooks (https://jupyter.org/)

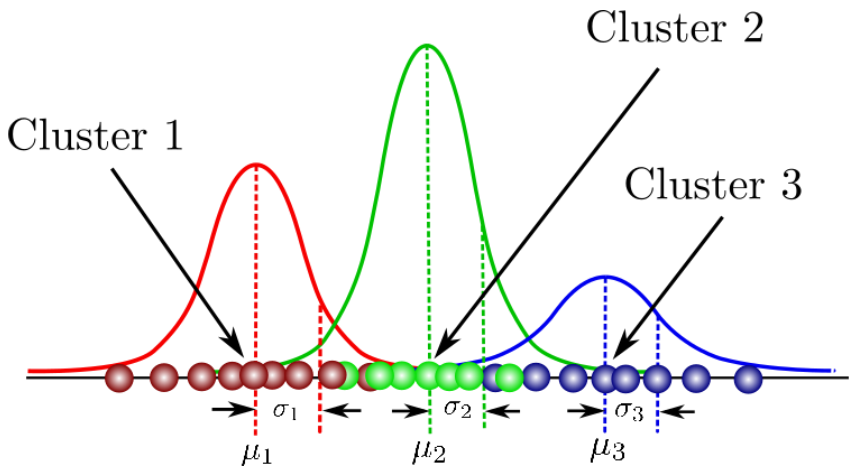

Imagen 5. Ejemplo de un modelo de clusterización Gaussian Mixture de 3 clústeres

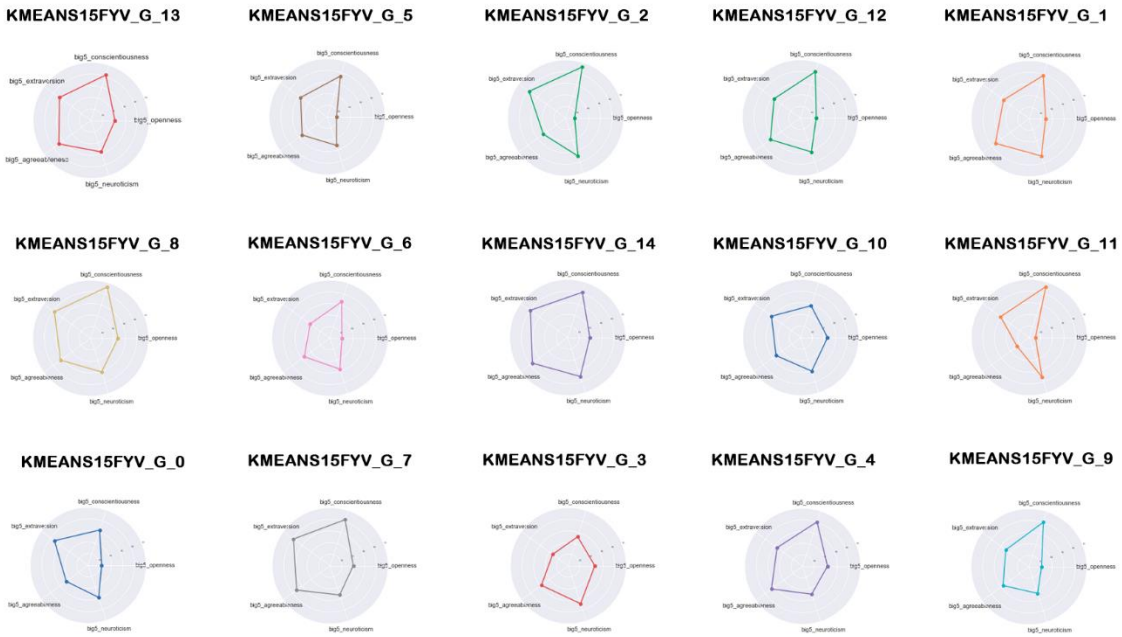

Imagen 6. Comparativa de los clústeres del modelo Gaussiano desarrollado sobre los Big-5

2.2.Alcance

Los objetivos a conseguir con esta iteración son los siguientes:

- Como se define en los requisitos, el desarrollo de un servicio web para interactuar con los diferentes modelos de clusterización.
- Desarrollo de un modelo de clusterización, Gaussian Mixture.
- Desarrollo de un cliente para interactuar desde línea de comandos con el servicio.
- Generar la documentación para la iteración.

2.3.Captura de requisitos

La captura de requisitos se realizó a través de reuniones con el tutor de la empresa. (Ver anexo, Actas de reuniones). Estos se han dividido en los siguientes bloques: Requisitos funcionales, Requisitos no funcionales, Requisitos transversales y Exclusiones. La tabla constará de dos columnas, una para el **código** del requisito con el que se referirán los requisitos dentro del presente documento y otra para la **descripción** de este.

2.3.1. Requisitos funcionales

En la Tabla 11 se detallan los requisitos no funcionales de la iteración 2. Estos requisitos tendrán la siguiente forma, **RFC – XX (Requisitos funcionales Clustericación).**

Requisitos funcionales del servicio	
Código	**Descripción**
RFC -01	Crear un notebook Python para entrenar y analizar los modelos de clusterización general.
RFC -02	Crear un modelo de clusterización Gaussiano.
RFC -03	Crear una clase de carga para los modelos de clusterización.
RFC -04	Los modelos estarán disponibles en la URL /<nombre del modelo>, esta URL debe aceptar peticiones POST.
RFC -05	Creación de una clase con los distintos métodos que queremos que tenga nuestro servicio.
RFC -06	Crear una clase para la estandarización de los datos de entrada en formato JSON para poder ser consumidos por los modelos.
RFC -07	Creación de un cliente para interactuar con el servicio desde línea de comandos.
RFC -08	Creación de un servicio para las métricas de los modelos.
RFC -09	Creación de una clase con las métricas para los diferentes métodos de los distintos servicios publicados.
RFC -10	Creación de una clase para el log de la API.
RFC -11	Los logs se deben de mostrar por pantalla, así como quedar registrados en un fichero de log.
RFC -12	Todos los métodos solo deben aceptar peticiones POST, salvo el método para la obtención de las métricas.

Tabla 11. Requisitos funcionales iteración 2

RFC -01 - Crear un notebook Python para entrenar y analizar los modelos de clusterización general:

Se creará un notebook de Jupyter en el cual se crearán los métodos necesarios para la carga de los ficheros de datos, entrenar los modelos y sacar un estudio de los clústeres obtenidos con ese modelo. La salida de este notebook será el informe que analizará el psicólogo para validar la coherencia de los mismos.

RFC -02 - Crear un modelo de clusterización Gaussiano:

Se desarrollará un modelo Gaussiano para la segmentación de los diferentes tipos de personas empleando como datos de entrada la salida del modelo Emocional desarrollado en la iteración 1 y que se encuentra ya publicado en la API.

RFC -03 - Creación de una clase con los diferentes métodos que queramos que tenga nuestro servicio:

Se creará una clase con los métodos para procesar las peticiones a las diferentes URL establecidas, estos métodos harán uso del modelo de clusterización previamente inicializado.

RFC -04 - Los modelos estarán disponibles en la URL /<nombre del modelo>, esta URL debe aceptar peticiones POST:

Este será el único método para los servicios de clusterización, el cual aceptará un entero o una lista de enteros. Estos datos se le pasará en el BODY de la petición.

RFC -05 - Creación de una clase con los distintos métodos que queremos que tenga nuestro servicio de clusterización:

Se desarrollará una clase con los diferentes métodos que queramos que tenga nuestro servicio web. Cada uno de estos métodos estará asociado a una URL con la cuál podremos interactuar con los modelos que se hubieran cargado previamente, por el momento solo estará cargado el modelo Gaussiano desarrollado. A estos métodos se les pasará como parámetros el nombre del modelo de clusterización que se desea emplear y el JSON de salida del modelo Emocional desarrollado en la iteración 1, y devolverán el entero del clúster que corresponda con la salida del modelo Emocional introducida.

RFC -06 - Crear una clase para la conversión de los datos de entrada en formato JSON para poder ser consumidos por los modelos:

Los métodos aceptarán la información en formato JSON, ya que esta información de entrada es la salida de los modelos Emocional. Esta clase se encargará de extraer los datos necesarios y crear el vector de entrada para los modelos de clusterización.

RFC -07 - Creación de un cliente para interactuar con el servicio desde línea de comandos:

Añadir al cliente desarrollado en la componente anterior los métodos necesarios para interactuar con los nuevos métodos. Al ejecutar este cliente en una consola nos aparecerá un menú en el cual podremos elegir el tipo de petición que deseemos realizar. Nos pedirá los datos de entrada que sean precisos y nos mostrará la salida por la consola.

RFC -08 - Creación de un servicio para las métricas de los modelos:

Se desarrollará un servicio web el cuál devolverá diferentes métricas de los métodos publicados en la API las cuales permitirán evaluar el uso de los diferentes métodos. Se publicará un método el cual nos devolverá todas las métricas en formato JSON.

RFC -09 - Creación de una clase con las métricas para los diferentes métodos de los distintos servicios publicados:

Se desarrollará una clase con diferentes métricas para el control y análisis de los métodos de la API. Las métricas a desarrollar son: el ratio de peticiones a los métodos en el tiempo (ratio en 1, 5 y 15 minutos), los percentiles (73, 95, 99, 999), el número de peticiones, la media y la desviación estándar.

RFC -10 - Creación de una clase para el log de la API:

Se creará una clase para el log personalizado de la API. Este log debe de funcionar correctamente independientemente del servidor en el cual se despliegue la API. Debe de ser personalizable mediante un fichero de configuración en formato JSON donde se establezcan todos los parámetros que deseemos.

RFC -11 - Los logs se deben de mostrar por pantalla, así como quedar registrado en un fichero de log:

Independientemente del servidor en el cual se despliegue la API y de la configuración que establezcamos, debe de existir la posibilidad de mostrar los mensajes de log en consola, así como crear un fichero de log donde se almacenen las llamadas realizadas a la API y se registren de forma permanente. La forma de mostrarse o de almacenarse en los ficheros será la que se indique en el fichero de configuración.

RFC -12 - Todos los métodos solo deben aceptar peticiones POST, salvo el método para la obtención de las métricas:

Puesto que se envía información en las peticiones a los métodos, estos deben de aceptar únicamente peticiones Post, a excepción del método de métricas que no recibe ningún parámetro y por lo tanto solo aceptará peticiones GET.

2.3.2. Requisitos no funcionales

En la Tabla 12 de detallan los requisitos no funcionales de la iteración 2. Estos requisitos tendrán la siguiente forma, **RNC – XX (Requisitos no funcionales Clusterización)**

Requisitos no funcionales del servicio	
Código	**Descripción**
RNC -01	Entrega del servicio de Clusterización el 10/04/2020.
RNC -02	Uso de Python para desarrollo del servicio Web.
RNC -03	Uso de Flask como servidor web.
RNC -04	Uso de Flasgger para documentar los métodos del servicio.
RNC -05	Los servicios deben ser multihilo.
RNC -06	Validación del modelo de clusterización.

Tabla 12. Requisitos no funcionales iteración 2

RNC -01 - Entrega del servicio de Clusterización el 10/04/2020:

La documentación generada acerca del servicio tiene que ser revisada y validada por el tutor de la empresa.

RNC -02 - Uso de Python para desarrollo del servicio Web:

Emplear como lenguaje de programación Python, ya que los modelos que se van a desarrollar y publicar estarán desarrollados con este lenguaje. Es por esto que lo empleamos, para que su integración con la API sea más simple.

RNC -03 - Uso de Flask como servidor web:

Emplear Flask como framework de desarrollo para crear los servicios web.

RNC -04 - Uso de Flasgger para documentar los métodos del servicio:

Emplear Flasgger para documentar lo métodos del servicio además de ofrecer una forma desde la propia API de interactuar con los métodos de los servicios publicados en la API.

RNC -05 - Los servicios deben ser multihilo:

Los servicios deben de ser multihilo, es decir que acepten peticiones concurrentes. Esto es necesario ya que cuando se suba a producción va a necesitar hilos para procesar de forma concurrente todas las peticiones y evitar de esta forma cuellos de botella en el servidor

RNC -06 – Validación del modelo de clusterización:

Pese a que el psicólogo de la empresa participará en la creación del modelo a desarrollar, este ha de ser validado por el mismo. La salida del modelo de clusterización debe de tener una cierta coherencia con respecto a las teorías psicológicas vigentes. Se busca que el modelo tenga una base científica.

2.3.3. Requisitos transversales

En la Tabla 13 de detallan los requisitos transversales de la iteración 2. Estos requisitos tendrán la siguiente forma, **RTC – XX (Requisitos transversales Clusterización)**

Requisitos transversales del servicio	
Código	**Descripción**
RTC -01	Validación del servicio por el tutor de la empresa.
RTC -02	Validación de la documentación del servicio por el tutor de la empresa.

Tabla 13. Requisitos transversales iteración 2

RTC -01 - Validación del servicio por el tutor de la empresa:

El servicio desarrollado tiene que ser revisado y aceptado por el tutor de la empresa.

RTC -02 - Validación de la documentación del servicio por el tutor de la empresa:

La documentación generada acerca del servicio tiene que ser revisada y validada por el tutor de la empresa.

2.3.4. Exclusiones

En la Tabla 14 de detallan los requisitos excluidos de la iteración 2. Estos requisitos tendrán la siguiente forma, **EC – XX (Exclusiones Clusterización)**

Exclusiones del servicio	
Código	**Descripción**
EC -01	Carga dinámica de los modelos de clusterización.
EC -02	Creación de Dockerfile.

Tabla 14. Exclusiones iteración 2

EC -01 - Carga dinámica de los modelos de clusterización:

La carga dinámica de los modelos nos permitiría cargar diferentes modelos sin necesidad de desarrollar nuevas clases. Sería necesario establecer una serie de parámetros en un fichero de configuración y al arrancar el servicio y los diferentes modelos se cargarían en tiempo de ejecución. Esto haría más simple el ir ampliando la API con nuevos modelos y servicios sin demasiado esfuerzo.

Está excluida la carga dinámica de los modelos puesto que no cumpliríamos con los plazos de entrega, se plantea como parte de una nueva iteración.

EC -02 - Creación de Dockerfile:

Con el servidor en local es suficiente para realizar las pruebas. Por el momento no será necesario, se creará cuando se vaya a subir a producción la aplicación de servicios.

2.4. Diseño

En esta iteración desarrollaremos un modelo de clusterización Gaussiano empleando el modelo sklearn.mixture.GaussianMixture. Lo ajustaremos para obtener el mejor modelo. Una vez obtenido el mejor modelo generamos un informe el cuál fue revisado y validado por el psicólogo de la empresa cumpliendo así el requisito RNC-06 definido previamente. Este modelo será el que carguemos en la API.

También añadiremos nuevos servicios a la API que comenzamos a desarrollar en la iteración 1. Tendremos una serie de servicios los cuales tendrán un método que estará asociado a una URL con la cual podremos invocarlos. En la Imagen 7 se muestra la estructura de las nuevas URL. Todas estas URLs cuelgan de dos URLs base:

/: Desde esta URL base colgarán los métodos para los modelos de clusterización publicados. Por el momento solo estará publicado el modelo Gaussiano que hemos desarrollado en esta iteración.

/<string:método>: Esta URL solo recibe peticiones POST, la cual acepta JSON en el BODY de la petición. Este JSON corresponde con la salida del modelo Emocional publicado en la componente anterior. Devuelve el entero asignado al clúster que se corresponde con la personalidad predicha de esa persona.

/metrics: Desde esta URL base colgará el método para la obtención de las métricas de todos los métodos publicados mediante URL.

/: Esta URL solo recibe peticiones GET, la cual al ser una petición GET no recibe ningún parámetro. Devuelve un JSON el cual contiene todas las métricas desarrolladas en la clase de métricas de los métodos que hayan sido invocados en algún momento.

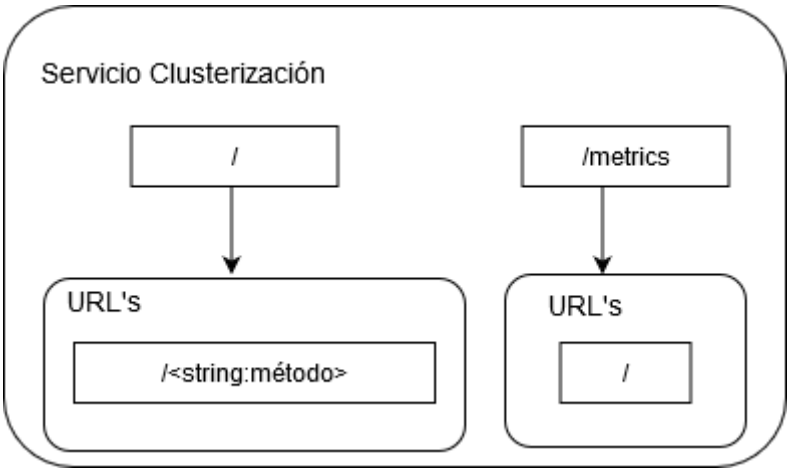

Imagen 7. Diagrama del diseño de los servicios de la iteración 2

2.5. Tecnologías

Para esta iteración se han empleado las siguientes tecnologías:

1. **Python** es el lenguaje con el cual desarrollaremos el servicio, habiéndolo elegido ya que el modelo Emocional está desarrollado con este mismo lenguaje.
2. **Flask** es un servidor web disponible en Python, con él lanzamos nuestro servicio.
3. **Flasgger** lo implementaremos para la autodocumentación de la API, permitiéndonos desde la propia API consultar los servicios disponibles e invocarlos sin necesidad de cliente.
4. **Blueprint** esta librería la usaremos para organizar la estructura interna del servicio, permitiéndonos crear agrupaciones de URLs según directorios. Con ello podemos establecer una URL base diferente para cada servicio web.
5. **Joblib** esta librería se empleará para serializar los modelos de clusterización que desarrollemos, por el momento solo será uno. De esta forma podremos exportar los modelos generados ya entrenados y así poder guardarlos y usarlos en nuestro servicio. Esta misma librería nos permite cargarlos también para poder usarlos. Elegí esta librería frente a otras ya que esta nos permitía comprimir los objetos serializados para reducir su tamaño si son muy grandes de forma sencilla y nativa.
6. **sklearn.mixture.GaussianMixture,** esta clase la emplearemos para ellos modelo Gaussiano. Es parte de la biblioteca de aprendizaje automático Scikit-learn[13].
7. **Pyformance**[14] es una librería Python con un conjunto de herramientas para medir el rendimiento y las estadísticas de los métodos como son: Counter (Interfaz simple para aumentar y disminuir un valor. Por ejemplo, esto se puede usar para medir el número total de peticiones recibidas por un método), Meter (Mide la tasa de eventos a lo largo del tiempo. Útil para realizar un seguimiento de la frecuencia con la que un método recibe peticiones a lo largo del tiempo) e Histogram (Mide la distribución estadística de valores en un flujo de datos. Realiza un seguimiento de las desviaciones mínimas, máximas, medias, estándar, etc. También mide los percentiles mediana, 75, 90, 95, 98, 99 y 99,9).

2.6. Implementación

El servicio web para la clusterización y el servicio para la obtención de las métricas se añaden a la API desarrollada en la iteración 1, la estructura se puede ver en la Imagen 8. Sobre la estructura de la iteración anterior se añadirán una serie de carpetas dentro de la carpeta **modelos** en la cual tendremos la nueva carpeta **modeloClusterización** donde tenemos todos

[13] **Scikit-learn:** es una biblioteca para aprendizaje automático de software libre para el lenguaje de programación Python. (https://scikit-learn.org/stable/)
[14] **Pyformance:** es una librería Python para el control del rendimiento y las estadísticas de los métodos. (https://pyformance.readthedocs.io/en/latest/)

los ficheros necesarios para desplegar el servicio que publicará los modelos de clusterización que vayamos desarrollando, aunque por el momento solo tendremos uno publicado, y los métodos desarrollados que tendremos asociados a las URLs que publiquemos. En **aplicacion/modelos/modelosClusterizacion/__init__.py** es donde mapeamos los métodos definidos en el fichero **aplicacion/modelos/modelosClusterizacion/baseRun.py.** En el fichero **aplicacion/modelos/modelosClusterizacion/cluster.py** se definen los métodos necesarios para la carga de los modelos serializados mediante la librería JobLib, además de los métodos para la conversión de los datos para poder ser consumidos por los modelos. En la carpeta **aplicacion/modelos/modelosClusterizacion/entrenados** será donde almacenemos los modelos ya entrenados.

El servicio para obtener la métrica se encuentra en la carpeta **aplicacion/metricas** donde tenemos el fichero **aplicacion/metricas/metricsRun.py** en el cual tenemos los métodos que queremos publicar, en nuestro caso solo uno.

También tenemos una carpeta con las clases de funcionalidades como la clase de las métricas (**aplicación/utils/monitor.py**) o la clase con el loggin personalizado para cuando despleguemos el servidor (**aplicación/utils/myloggin.py**).

Cuando por ejemplo hacemos una petición a la URL" http://localhost:5000/gauss ", queremos realizar una petición al método del modelo de clusterización Gaussiano, como se puede ver en la URL. Dicha petición debe de ser POST y en el cuerpo de la petición (body) debe obtener la salida JSON de uno de los métodos Emocionales. El servidor recibirá esta petición y mediante la tecnología *blueprint* haremos llegar la petición con el método asociado con esa URL. Este método obtendrá el JSON del body de la petición y llamará el método que transformará el JSON recibido para poder ser consumido por el método de clusterización y finalmente llamará al modelo de clusterización, en este caso Gaussiano, y devolverá el número del cluster al que correspondan los datos de entrada recibidos.

Si hacemos una petición a la URL" http://localhost:5000/metrics/ ", queremos obtener las métricas y estadísticas de los modelos que han recibido peticiones. Dicha petición ha de ser GET.

Este método devolverá un JSON con todas las métricas disponibles sin necesidad de transformación ya que Flask permite devolver JSON sin necesidad de serialización. Un ejemplo de salida lo tenemos en el Anexo, Ejemplo de las salidas de las métricas.

Toda petición que llega a nuestra API es registrada o bien un fichero de configuración, por consola o ambas, según se configure. Esto es posible ya que hemos establecido un evento para que toda respuesta que se devuelva desde cualquiera de los métodos publicados por la API sea

procesada antes de ser devuelta permitiendo extraer toda la información de la petición y de la respuesta sin necesidad de tratar desde cada uno de los métodos la gestión del registro de los mensajes en el log, haciendo que haya menos código repetido en los métodos y por tanto una mayor simplicidad en el código. Tendremos 3 niveles: Info para cuando todo vaya bien (Por ejemplo, código de respuesta 200), Warning cuando se produzca, por ejemplo, una redirección y Error cuando se produzca cualquier tipo de error por cualquier motivo. Se puede ver la salida por consola en la Imagen 9.

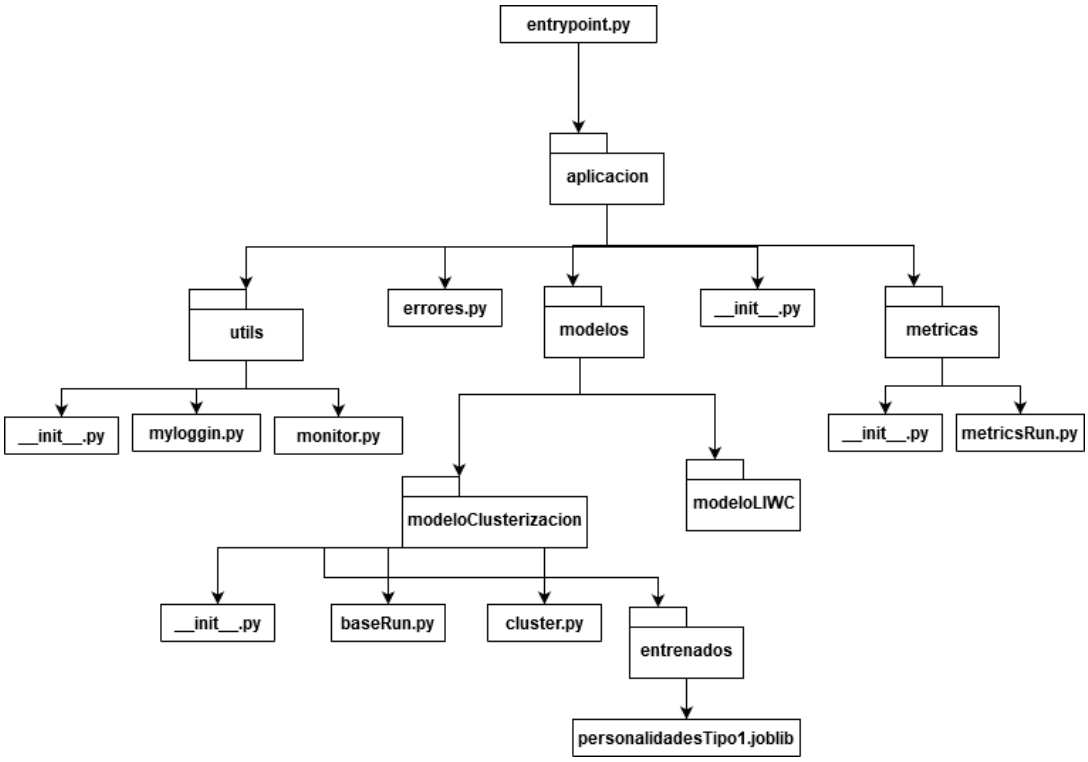

Imagen 8. Estructura de la aplicación web con la iteración 2

```
2020-06-03 20:18:34.936727: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that thi
s TensorFlow binary was not compiled to use: AVX2
C:\Users\Millan\Desktop\Webtdv\env\lib\site-packages\sklearn\base.py:334: UserWarning: Trying to unpickle estimator Gaus
sianMixture from version 0.22.2.post1 when using version 0.23.0. This might lead to breaking code or invalid results. Us
e at your own risk.
  UserWarning)
2020-06-03 20:18:53,916 - TDV - INFO - Iniciando servidor host:localhost puerto:5000
 * Serving Flask app "webservS.aplicacion" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
2020-06-03 20:21:29,928 - TDV - WARNING - 127.0.0.1 - RESPONSE:308 PERMANENT REDIRECT -- URL:http://localhost:5000/apido
cs
2020-06-03 20:21:30,038 - TDV - INFO - 127.0.0.1 - RESPONSE:200 OK -- URL:http://localhost:5000/apidocs/
2020-06-03 20:21:30,174 - TDV - INFO - 127.0.0.1 - RESPONSE:200 OK -- URL:http://localhost:5000/flasgger_static/swagger-
ui.css
2020-06-03 20:21:30,417 - TDV - INFO - 127.0.0.1 - RESPONSE:200 OK -- URL:http://localhost:5000/flasgger_static/swagger-
ui-bundle.js
2020-06-03 20:21:30,427 - TDV - INFO - 127.0.0.1 - RESPONSE:200 OK -- URL:http://localhost:5000/flasgger_static/swagger-
ui-standalone-preset.js
2020-06-03 20:21:30,448 - TDV - INFO - 127.0.0.1 - RESPONSE:200 OK -- URL:http://localhost:5000/flasgger_static/lib/jque
ry.min.js
2020-06-03 20:21:31,386 - TDV - INFO - 127.0.0.1 - RESPONSE:200 OK -- URL:http://localhost:5000/flasgger_static/favicon-
32x32.png
2020-06-03 20:21:31,456 - TDV - INFO - 127.0.0.1 - RESPONSE:200 OK -- URL:http://localhost:5000/apispec_1.json
2020-06-03 20:21:47,596 - TDV - INFO - 127.0.0.1 - RESPONSE:200 OK -- URL:http://localhost:5000/liwc/es
2020-06-03 20:22:07,952 - TDV - INFO - 127.0.0.1 - RESPONSE:200 OK -- URL:http://localhost:5000/gauss
2020-06-03 20:22:24,289 - TDV - INFO - 127.0.0.1 - RESPONSE:200 OK -- URL:http://localhost:5000/metrics/
2020-06-03 20:23:53,930 - TDV - ERROR - 501 Not Implemented: The server does not support the action requested by the bro
wser.
2020-06-03 20:23:53,937 - TDV - ERROR - 127.0.0.1 - RESPONSE:501 NOT IMPLEMENTED -- URL:http://localhost:5000/gauss
```

Imagen 9. Salida por consola del log personalizado

2.7.Seguimiento y control

En este apartado analizaremos el desarrollo de la iteración, así como sus desviaciones con respecto a la planificación inicial.

2.7.1. Seguimiento de cambios

Durante el desarrollo de esta iteración no surgió ningún cambio, por lo que el desarrollo de la iteración no se ha visto afectado por ellos.

2.7.2. Seguimiento del alcance

Al comenzar la iteración se establecieron una serie de requisitos los cuales se iban a tratar de desarrollar a lo largo de la iteración, es por ello que debemos analizar la situación del alcance al finalizar la iteración.

2.7.3. Requisitos alcanzados

Se han cumplido satisfactoriamente todos los requisitos especificados en el apartado "2.2. Alcance".

2.7.4. Entregables generados

Se ha desarrollado el siguiente entregable:

- **RNC-1 Documentación:** contendrá las especificaciones del servicio web desarrollado en esta iteración, así como un manual de su uso e implementación del servicio.

2.8.Problemas

Durante la realización de esta componente no nos encontramos con ningún problema, por lo que no se ha sufrido ninguna clase de retraso en su desarrollo.

3. Iteración 3: Carga dinámica

3.1. Definición de la iteración

Esta será la iteración final, será un poco más extensa que las anteriores y por ello requerirá más tiempo. Añadiremos un servicio para obtener la metainformación de los modelos de clusterización publicados, rediseñaremos la estructura interna de la API para permitir la carga dinámica de los modelos a publicar y por último nos centraremos en la publicación en producción de la API.

Los modelos publicados pueden tener una metainformación asociada a ellos, por lo general información sobre la salida que producen. Por el momento el único de los modelos publicados que tendrá metainformación asociada será el modelo de clusterización Gaussiano, la cual será el nombre y la descripción de cada uno de los clústeres. Esta metainformación se publicará como un servicio más de nuestra API.

Una mejora para la API es la carga dinámica de los modelos a publicar en la API. La carga dinámica de los modelos nos permitirá elegir en un determinado momento qué modelos publicar o añadir nuevos modelos a nuestra API sin necesidad de escribir código nuevo, esto supone un ahorro de costes y de tiempo para la empresa. La carga dinámica cogerá la configuración de un fichero en el cuál estableceremos los modelos que queramos cargar, así como los parámetros necesarios para su carga. Por el momento la carga dinámica solo cargará los modelos ya desarrollados en las iteraciones anteriores, el modelo Emocional y el modelo de clusterización desarrollado en la componente 2. Los modelos a cargar podrán estar en dos fuentes diferentes, permitiendo cargar modelos que se encuentren en local o en remoto, nosotros utilizaremos la tecnología de Azure Blob Storage[15]. Los modelos almacenados en remoto contarán con los mismos archivos que cuando se carguen desde local, la única diferencia es la fuente de donde obtenemos los modelos a cargar. Un esquema del concepto de carga dinámica se puede ver en la Imagen 10.

Por último, se subirá la API a producción. Para ello es necesario elegir un servidor que se adecue a nuestras necesidades y así sustituir el servidor de desarrollo que hemos estado usando hasta el momento, ya que este no puede ser usado en el momento que la API se suba a producción. Tras elegir el servidor más adecuado para lanzar la API a producción, generaremos la imagen Docker que nos permita publicarla. Esta imagen se subirá a Azure Container Registry[16] para que pueda ser utilizada por toda la empresa sin necesidad de regenerarla cada vez que sea necesaria. Una vez publicadas las imágenes Docker con la API emplearemos Kubernetes[17] para

[15] **Azure Blob Storage:** es la solución de almacenamiento de objetos de Microsoft para la nube. (https://azure.microsoft.com/es-es/services/storage/blobs/)

[16] **Azure Container Registry:** es un servicio privado administrado del Registro de Docker donde se puede crear y mantener los registros de Azure Container para almacenar y administrar las imágenes privadas de contenedores Docker y sus artefactos relacionados. (https://azure.microsoft.com/es-es/services/container-registry/)

[17] **Kubernetes:** es una plataforma portable y extensible de código abierto para administrar cargas de trabajo y servicios. Kubernetes facilita la automatización y la configuración declarativa. (https://kubernetes.io/es/)

balancear la carga y gestionar los Pods[18] Docker. En la Imagen 11 se puede ver un esquema de una arquitectura Docker-Kubernetes.

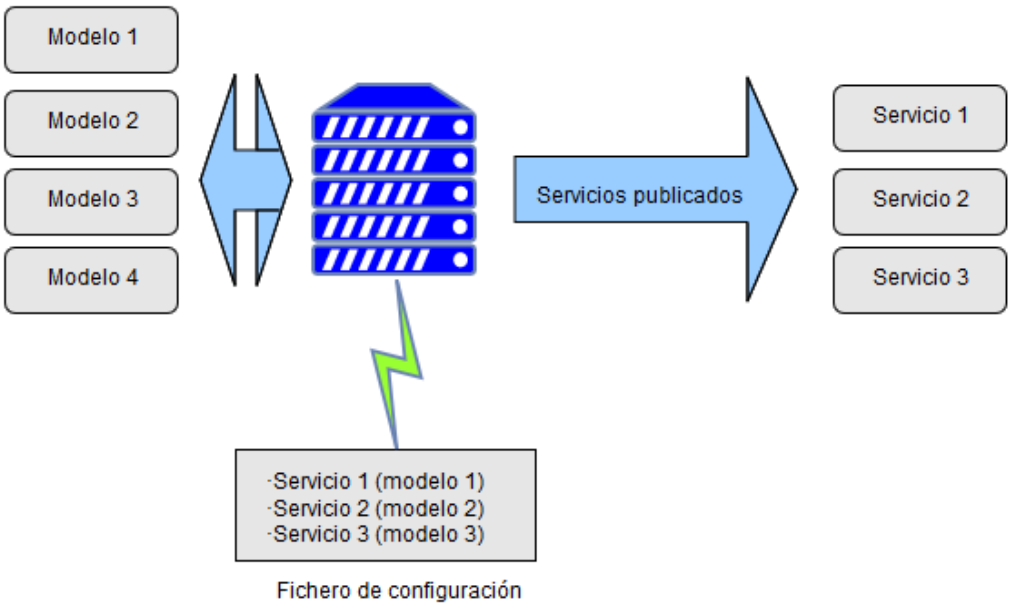

Imagen 10. Diagrama conceptual de la carga dinámica de modelos

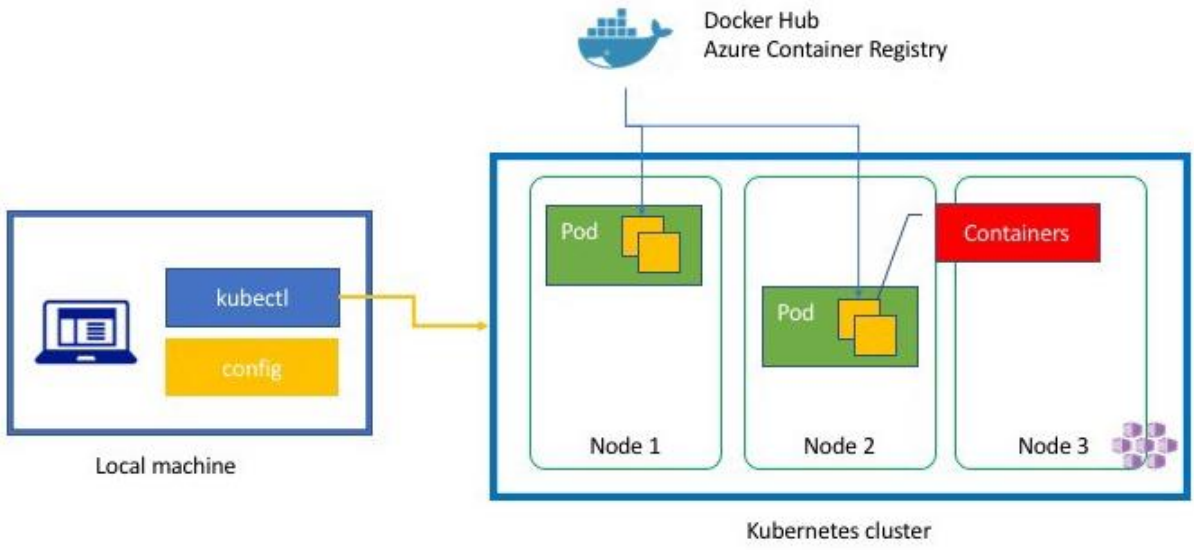

Imagen 11. Esquema de una arquitectura Docker-Kubernetes

[18] **Pod:** es un grupo de uno o más contenedores Docker con almacenamiento o red compartidos y una especificación sobre como ejecutarlos.

3.2. Alcance

Los objetivos a conseguir con esta iteración son los siguientes:

- Como se define en los requisitos, el desarrollo de un servicio web para obtener la metainformación de los modelos publicados.
- Diseño de la carga dinámica de los modelos.
- Dockerización de la API.
- Publicación de la API mediante Docker-Kubernetes.
- Generar la documentación de la iteración y el despliegue de la API.

3.3. Captura de requisitos

La captura de requisitos se realizó a través de reuniones con el tutor de la empresa. (Ver Anexo, Actas de reuniones). Estos se han dividido en los siguientes bloques: Requisitos funcionales, Requisitos no funcionales, Requisitos transversales y Exclusiones. La tabla constará de dos columnas, una para el **código** del requisito con el que se referirán los requisitos dentro del presente documento y la **descripción** de este.

3.3.1. Requisitos funcionales

En la Tabla 15 de detallan los requisitos no funcionales de la iteración 3. Estos requisitos tendrán la siguiente forma, **RFD – XX (Requisitos funcionales dinámico).**

Requisitos funcionales del servicio	
Código	**Descripción**
RFD -01	Creación una clase de carga para la metainformación de los modelos.
RFD -02	Creación de una clase con los diferentes métodos que queramos que tenga nuestro servicio de metainformación.
RFD -03	Creación de un método en el servicio de metainformación para obtener la metainformación de un modelo, al cual se le pasan los datos de entrada en el body de la petición y el nombre del modelo.
RFD -04	Creación de la clase para la carga de la configuración desde un fichero de configuración.
RFD -05	Desarrollo de la carga dinámica de los modelos.
RFD -06	Creación del fichero de configuración para los modelos ya desarrollados en formato JSON.
RFD -07	La aplicación será capaz de cargar modelos que se encuentren tanto de forma local como remota.
RFD -08	Desarrollo del fichero Dockerfile.
RFD -09	Desarrollo de la descripción de un objeto de Kubernetes en formato YAML[19].
RFD -10	Desarrollo de fichero bash para lanzar el servidor.
RFD -11	Desarrollo de la carga dinámica de la metainformación.

Tabla 15. Requisitos funcionales iteración 3

[19] **YALM:** es un formato de serialización de datos legible por humanos inspirado en lenguajes como XML, C, Python, Perl, así como el formato para correos electrónicos especificado en RFC 2822. (https://yaml.org/)

RFD -01 - Creación una clase de carga para la metainformación de los modelos:

Se creará una clase a modo de factoría de recursos para cargar la metainformación de todos los modelos publicados. El concepto de factoría de recursos permite cargar diferentes recursos similares para así tenerlos agrupados en un único sitio siendo accesibles en toda la aplicación. De esta forma los recursos solo se instancian una única vez. Con la factoría de metainformación ya cargada y sabiendo el modelo del cual queremos obtener la metainformación, solo tendremos que obtener el diccionario de la factoría y extraer el fragmento que nos interese.

RFD -02 - Creación de una clase con los diferentes métodos que queramos que tenga nuestro servicio de metainformación:

Se creará una clase con los métodos para procesar las peticiones a las diferentes URL establecidas, estos métodos harán uso de la factoría de metainformación previamente inicializada.

RFD -03 - Creación de un método en el servicio de metainformación para obtener la metainformación de un modelo, al cual se le pasan los datos de entrada en el body de la petición y el nombre del modelo:

Este método del servicio web será invocado mediante una petición Post a su correspondiente URL. En dicha petición POST, el body de la petición podrá contener diferentes datos de entrada: si no se le pasa ningún dato se devolverá el fichero completo, una clave para devolver una sección del fichero en formato JSON o una lista de claves para devolver una lista de secciones del fichero en formato JSON.

RFD -04 - Creación de la clase para la carga de la configuración desde un fichero de configuración:

Se desarrollará una clase para cargar la configuración de la API desde un fichero JSON con los diferentes parámetros que deseemos cargar en el servidor. Esta clase una vez cargada la configuración nos permitirá obtenerla sin necesidad de leerla cada vez que la necesitemos desde cualquier archivo de la API.

RFD -05 – Desarrollo de la carga dinámica de los modelos:

Se diseñará una nueva estructura para el backend de la API para permitir la carga en tiempo de ejecución de los modelos. Para la carga dinámica desarrollaremos las clases de carga necesarias, una para los modelos de clusterización y otra para los modelos Emocionales, estas clases harán uso de la clase de carga de la configuración. Ambas clases hará uso del concepto de factoría de recursos en este caso de modelos para que los modelos estén disponibles para su uso.

RFD -06 – Creación del fichero de configuración para los modelos ya desarrollados en formato JSON:

Se diseñará una estructura JSON para establecer los parámetros de configuración para la carga de los modelos, además de la configuración del log. Se pretende almacenar toda la información de las configuraciones en un único dichero que se cargará al iniciar el servidor. Este fichero una vez diseñado se debe de rellenar con los parámetros necesarios para su carga local, es decir que ya se encuentren todos los archivos y modelos necesarios en local.

RFD -07 – La aplicación será capaz de cargar modelos que se encuentren tanto de forma local como remota:

Desarrollo de una interfaz para la obtención de los modelos. Esta interfaz se implementará para la carga de los modelos que se encuentren en local y para los modelos que se encuentren en un repositorio remoto y sea necesario descargarlos. La implementación de la obtención dinámica

de los modelos se hará teniendo en cuenta que los modelos estarán almacenados en Azure Blob Storage.

RFD -08 – Desarrollo del fichero Dockerfile:

Desarrollo del fichero DockerFile para crear la imagen Docker necesaria para lanzar el servidor con esta tecnología. En él se establecerá la imagen base de la que partir la cual definirá el sistema operativo, además de configurar los puertos publicados y el comando que se ejecutará al iniciar la imagen.

RFD -09 – Desarrollo de la descripción de un objeto de Kubernetes en formato YAML:

Desarrollo de un fichero en formato YAML para definir los parámetros del despliegue en Kubernetes. Definir los Pods, las imágenes Docker, emplearemos la imagen docker que desarrollaremos y subiremos a Azure Container Registry, además de los puertos a publicar o la dirección URL única desde la que poder interactuar con ellos.

RFD -10 – Desarrollo de fichero bash para lanzar el servidor:

Desarrollo de un fichero bash que funcione a modo de comando que reciba parámetros. Con él podremos clonar o actualizar repositorios, añadir rutas al PYTHONPATH[20], instalar requirements (librerías Python) y lanzar el servidor en sistemas operativos Linux.

RFD -11 – Desarrollo de la carga dinámica de la metainformación:

Al igual que con los modelos, se debe de desarrollar las clases necesarias para la carga dinámica de la metainformación, que se podrá obtener de local o de remoto al igual que los modelos. Se debe hacer uso de la factoría desarrollada.

3.3.2. Requisitos no funcionales

En la Tabla 16 de detallan los requisitos no funcionales de la iteración 3. Estos requisitos tendrán la siguiente forma, **RND – XX (Requisitos no funcionales dinámico)**

RND -01 - Entrega del servicio de la iteración3 el 08/05/2020:

Entrega de la documentación generada acerca la iteración, así como lo desarrollado en ella.

RND -02 - Uso de Python para desarrollo del servicio Web:

Se usará Python para desarrollar el servicio de metainformación y la carga dinámica ya que es el lenguaje en el que está desarrollada toda la API.

RND -03 - Uso de un servidor web de producción:

No se puede emplear el mismo servidor para desarrollar que para producción, es por ello que habrá que buscar un servidor para producción que se adecue a nuestras necesidades.

RND -04 - Uso de Flasgger para documentar los métodos del servicio:

Emplear Flasgger para documentar lo métodos del servicio.

RND -05 – Todos los servicios publicados deben ser multihilo:

Todos los servicios deben de ser multihilo, es decir que acepten peticiones concurrentes. Para que esto sea posible la carga dinámica tiene que permitir la concurrencia de acceso a los recursos disponibles en las factorías.

[20] **PYTHONPATH:** es una variable de entorno que puede configurar para agregar directorios adicionales donde python buscará módulos y paquetes.

RND -06 - Todos los métodos solo deben aceptar peticiones POST:

Puesto que se envía información en las peticiones a los métodos, estos deben de aceptar únicamente peticiones Post.

RND -07 - El fichero de configuración de la carga dinámica debe de ser en formato JSON:

La empresa ha especificado que quiere que el fichero con la configuración de la carga dinámica de los modelos debe de estar en formato JSON.

RND -08 - Diseño de la estructura de almacenamiento de los modelos en Azure Blob Storage:

Se debe diseñar una estructura jerárquica para almacenar los modelos desarrollados en la tecnología de Azure Blob Storage. Esta estructura debe de tener una estructuración lógica permitiendo diferenciar por tipo de modelo además de por idioma, aunque no siempre será necesario ya que algunos modelos no son de ningún idioma en especial.

RND -09 - La configuración de los modelos y la del log deben de estar en el mismo fichero de configuración:

Por petición de la empresa, solo debemos tener un único fichero de configuración para toda la API. Se quiere tener en un único fichero toda la configuración para así evitar problemas de complejidad con los ficheros a la hora de configurarla.

Requisitos no funcionales del servicio	
Código	**Descripción**
RND -01	Entrega de la iteración 3 el 08/05/2020.
RND -02	Uso de Python para desarrollo del servicio Web.
RND -03	Uso de un servidor web de producción.
RND -04	Uso de Flasgger para documentar los métodos del servicio.
RND -05	Todos los servicios publicados deben ser multihilo.
RND -06	Todos los métodos solo deben aceptar peticiones POST.
RND -07	El fichero de configuración de la carga dinámica debe de ser en formato JSON.
RND -08	Diseño de la estructura de almacenamiento de los modelos en Azure Blob Storage.
RND -09	La configuración de los modelos y la del log deben de estar en el mismo fichero de configuración.

Tabla 16. Requisitos no funcionales iteración 3

3.3.3. Requisitos transversales

En la Tabla 17 de detallan los requisitos transversales de la iteración 3. Estos requisitos tendrán la siguiente forma, **RTD – XX (Requisitos transversales dinámico):**

RTD -01 - Validación de lo desarrollado en la iteración por el tutor de la empresa:

Todo lo desarrollado tiene que ser revisado y aceptado por el tutor de la empresa.

RTD -02 - Validación de la documentación del servicio por el tutor de la empresa:

La documentación generada acerca del servicio tiene que ser revisada y validada por el tutor de la empresa.

RTD -03 – Publicación de la API:

Al finalizar la iteración la API debe estar publicada en producción para poder ser ya usada por la empresa. Pero solo si es validada por el tutor de la empresa, RTD-01.

Requisitos transversales del servicio	
Código	**Descripción**
RTD -01	Validación de lo desarrollado en la iteración por el tutor de la empresa.
RTD -02	Validación de la documentación del servicio por el tutor de la empresa.
RTD -03	Publicación de la API.

Tabla 17. Requisitos transversales iteración 3

3.3.4. Exclusiones

En esta iteración no habrá ninguna exclusión ya que se terminará por completo el desarrollo de la API.

3.4. Diseño

En esta iteración añadiremos el servicio para obtener la metainformación de los modelos de clusterización a la API que comenzamos a desarrollar en la iteración 1. En dicho servicio tendremos un método, el cual será invocado desde la URL que tenga asociado. Este método colgará de la URL base "/meta". En la Imagen 12 se puede ver la estructura de URLs de la API completa:

/<string:método>: Esta URL solo recibe peticiones POST, la cual acepta datos en el BODY de la petición y en la URL el nombre del modelo del cual queremos obtener la metainformación. Este método acepta tres tipos de entradas en el body: si no hay datos, si se le pasa un único dato o si se le pasa una lista de datos de entrada. Si no recibe datos de entrada devuelve el fichero de metainformación completo. Si solo recibe un único dato de entrada, este dato es una clave del fichero de metainformación, nos devolverá la sección correspondiente del fichero de metainformación. Y si lo que recibe es una lista de datos de entrada nos devuelve una lista con las secciones correspondientes.

En todas las iteraciones hemos estado empleando Flasgger para ir documentando los métodos de la API. En la Imagen 13 se puede ver un fragmento del resultado final. Se puede acceder a ella desde "/apidocs"

3.5. Tecnologías

Para esta iteración se han empleado las siguientes tecnologías:

1. **Python** es el lenguaje con el cual desarrollaremos el servicio y la carga dinámica, habiéndolo elegido ya que la API por completo está desarrollada con él.
2. **Flasgger** lo usaremos por el mismo motivo que en las anteriores iteraciones.
3. **Blueprint** esta librería la usaremos por el mismo motivo que en las anteriores iteraciones.

4. **Gunicorn**[21] es un servidor HTTP de producción escrito en Python el cual emplearemos en producción con la API.
5. **Docker** [22] lo emplearemos para desplegar la API en contenedores software sin necesidad de máquinas virtuales. Podremos generar varios Pods y orquestarlos con Kubernetes.
6. **Kubectl**[23] **es** la interfaz de línea de comando que emplearemos para ejecutar comandos sobre despliegues clusterizados de Kubernetes.
7. **Kubernetes** es un orquestador de contenedores que usaremos para crear despliegues automáticos todo lo complejos que necesitemos, con balanceado de carga automático y monitorización de salud.

3.6. Implementación

En esta iteración hemos diseñado la carga dinámica de los modelos y la metainformación. Para ello hemos tenido que reestructurar la organización de los ficheros para así hacer más simple la carga dinámica. La solución que hemos encontrado ha sido transformar nuestro proyecto en una librería llamada **tdvwebAPI** en la cual tendremos las clases necesarias para la publicación de los métodos en la carpeta **aplicacion**, y en la carpeta **configuracion** tendremos las clases necesarias para la carga dinámica de los modelos y la metainformación. En la carpeta **utils** tenemos utilidades para la carga de los modelos que lo necesiten, en este caso solo lo necesita el modelo Emocional. Tenemos los mismos ficheros que en las iteraciones anteriores pero organizados de una forma más lógica en forma de librería. La estructura completa de la librería de la API la podemos ver en la Imagen 14. Con esta estructura en forma de librería podremos añadir nuevos servicios y funcionalidades de forma simple, y estos cambios estarán disponibles para todos los recursos o proyectos que la empleen de forma automática con tan solo bajarnos los cambios haciendo más fácil el mantener estos proyectos con la última versión de la API.

El fichero de configuración lo definimos en dos bloques, uno para los modelos y otro para el log. El bloque de los modelos estará dividido en dos bloques, uno para los modelos de clusterización y otro para los modelos Emocionales, hacemos esta distinción porque estos dos tipos de modelos no tienen los mismos métodos. En él hemos definido los parámetros necesarios para descargarlos de internet u obtener los modelos de forma local. Además de la estructura de los ficheros de la API también hemos diseñado de forma lógica la estructura de carpetas de Azure Blob Storage donde guardaremos los modelos para poder obtenerlos mediante una URL. La carga dinámica carga los modelos, la metainformación de los modelos y la configuración de log de forma automática al iniciar la API. Para evitar tener que leer el fichero de configuración

[21] **Gunicorn:** es un servidor HTTP Python WSGI (La interfaz de puerta de enlace de servidor web es una convención de llamada simple para que los servidores web reenvíen solicitudes a aplicaciones web o marcos escritos en el lenguaje de programación Python.) para UNIX. (https://gunicorn.org/)
[22] **Docker:** es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos. (https://www.docker.com/)
[23] **Kubectl:** es una interfaz de línea de comandos para ejecutar comandos sobre despliegues clusterizados de Kubernetes. Esta interfaz es la manera estándar de comunicación con el clúster ya que permite realizar todo tipo de operaciones sobre el mismo.
(https://kubernetes.io/docs/reference/kubectl/kubectl/)

múltiples veces hemos desarrollado también una clase para almacenar la configuración y que se pueda acceder a ella sin necesidad de leerla cada vez que la necesitemos.

Añadiremos a la API un nuevo servicio para obtener la metainformación asociada a un modelo si este la tuviera. Esta metainformación se carga en tiempo de ejecución con la carga dinámica y se almacena en la factoría de metainformación para que esté disponible. Dispondremos de un método para obtener esta metainformación publicado en la API, el cual recibe tres tipos de entradas: no recibe ningún parámetro, un solo dato de entrada o una lista con datos de entrada. El comportamiento de este método está definido en el requisito RFD -03.

Para poder lanzar el servidor Gunicorn desde la imagen Docker que hemos desarrollado tenemos un fichero Bash, que emplearemos a modo de comando para la terminal, el cual nos permite lanzar el servidor cargando repositorios, añadiendo rutas al PYTHONPATH y estableciendo el puerto o la ip en la que publicar el servidor. En la Tabla 18 se pueden ver los diferentes parámetros que acepta este fichero Bash. Un ejemplo del uso del fichero Bash sería: "run -p /src/keops.digital/source/lib: -w 4 -i 0.0.0.0 -o 5000 -e webservS.aplicacion -v createapp() -t requirements.txt"

- p: Establecemos las rutas de búsqueda para los repositorios.
- w: Establecemos los workers (el número de procesos de trabajo) a 4.
- i: Establecemos la ip 0.0.0.0, de esta forma todos todas las IPv4 tienen acceso a la API.
- o: Establecemos el puerto 5000 para que se publique el servicio.
- e: Establecemos el fichero de entrada para el servicio. Dicho fichero sería webservS.aplicacion.
- v: Establecemos que la variable que se tiene que publicar del fichero webservS.aplicacion es la devuelta por la función createapp().
- t: Establecemos el fichero de requirements.txt desde el cual queremos que se instalen las dependencias.

Por último, hemos creado un fichero YAML con la configuración necesaria para lanzar Kubernetes y poder orquestar y controlar los Pods Docker con la API y poder usarla en producción. En él hemos establecido los Pods a levantar, el puerto en el que publicar y demás configuraciones. Este fichero se puede ver en el anexo, Fichero yalm para Kubernetes.

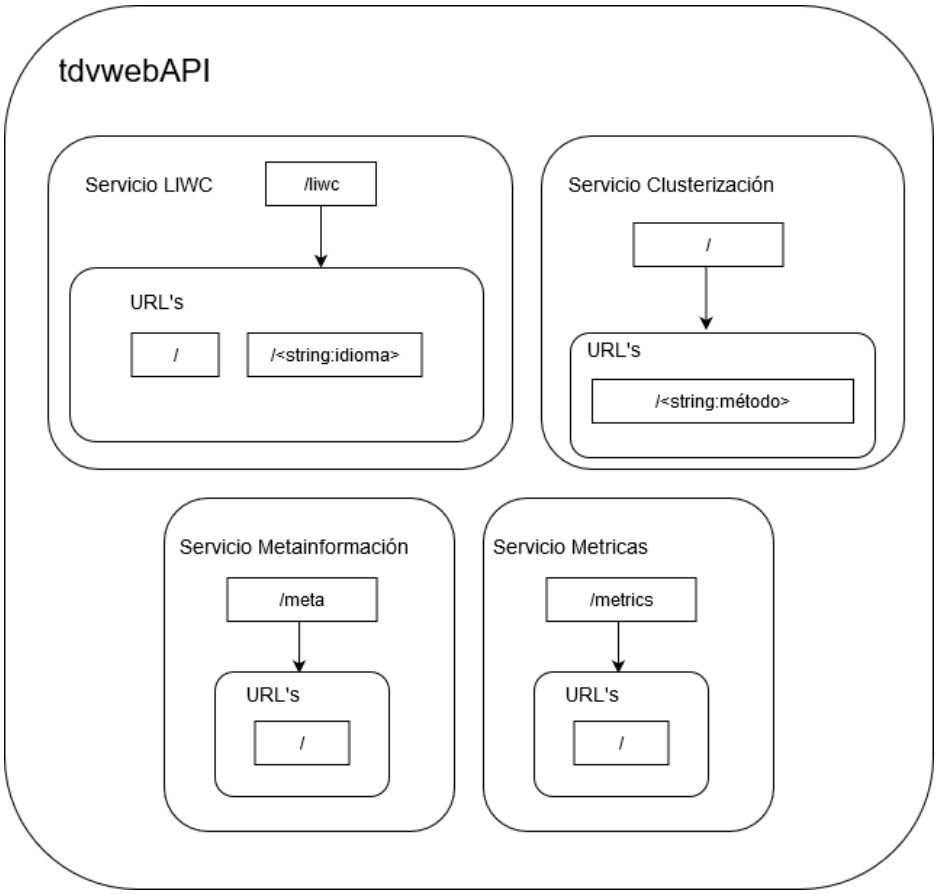

Imagen 12. Esquema del diseño de la API

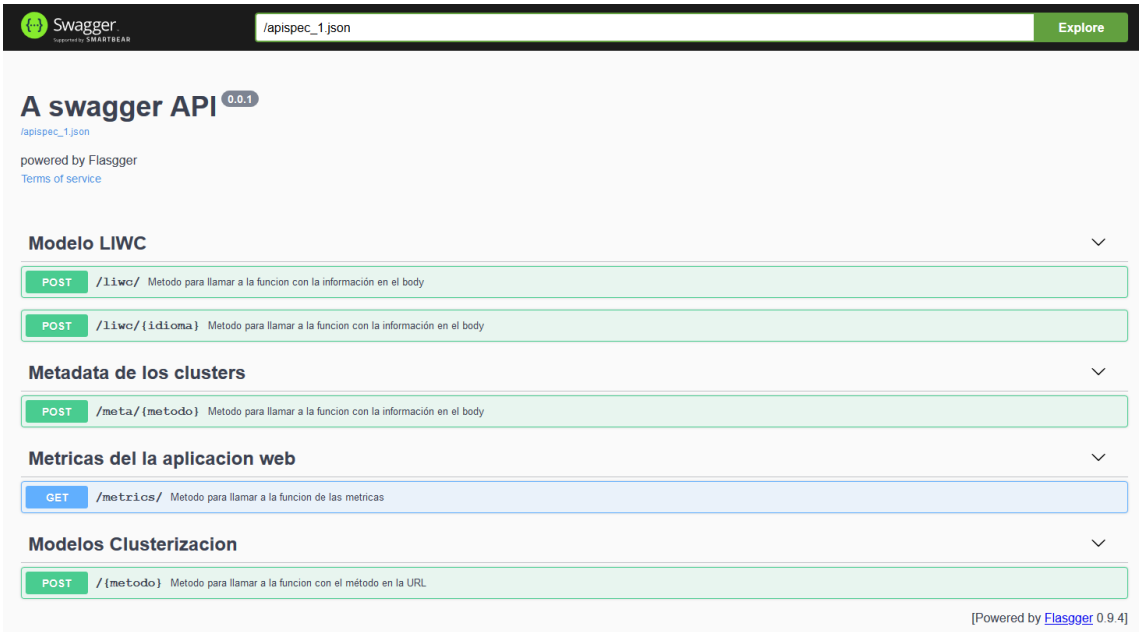

Imagen 13. Documentación con Flasgger

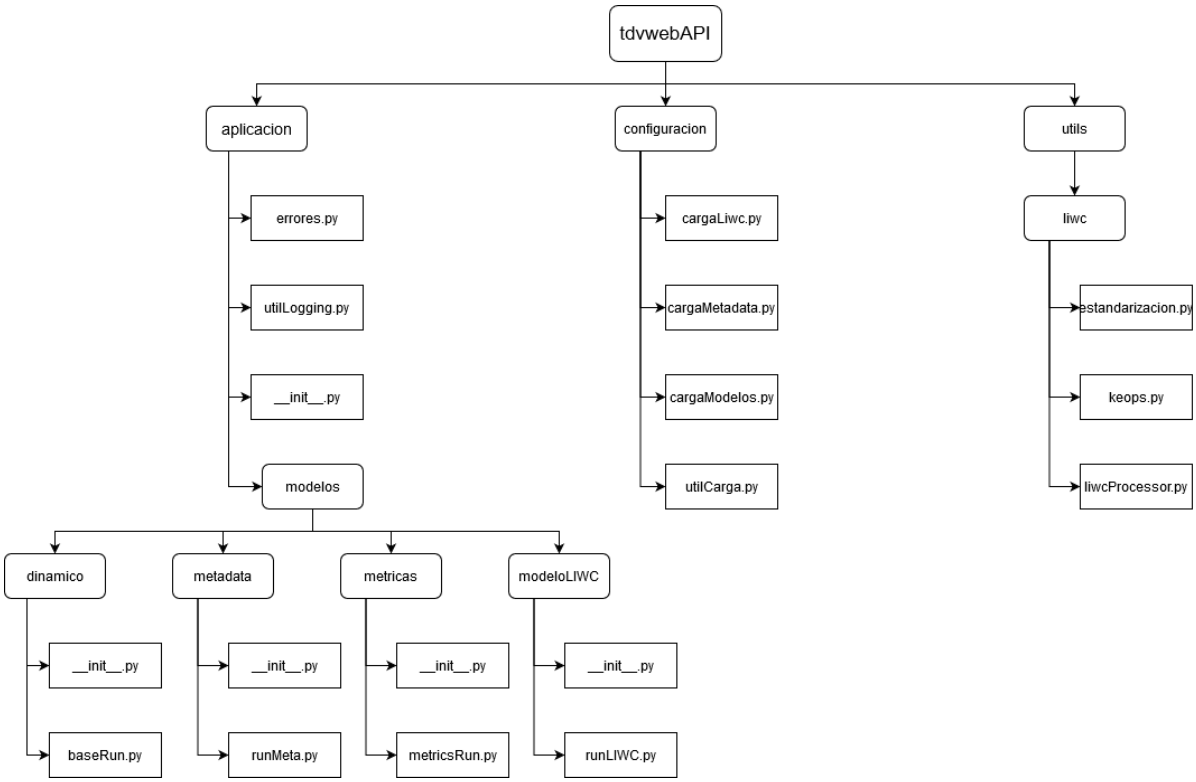

Imagen 14. Estructura de la API como librería

Parámetro	Descripción
-r <>	Los repositorios que se quieran descargar de git. Pueden ser uno o más de uno. Si es uno no hace falta meterlo entre comillas dobles. si queremos establecer la rama del repositorio hay que seguir la siguiente estructura: :. Si dicho repositorio ya está clonado bajará los últimos cambios. Estos cambios se descargarán en el directorio actual. Los credenciales por el momento deben de ser introducidos por consola.
-p <>	Las rutas de búsqueda. Toda ruta debe de terminar en ":". Por ejemplo una sola: -p /src/keops.digital /source/lib: o más de una -p /src/keops.digital/source/lib:/src/keops.digital/source/lib:/src/keops.digital/source /lib/webservS:
-w <>	Para establecer los Workers del comando de ejecución del servidor Gunicorn. Por defecto es 4.
-i <>	Para establecer la IP en la que publicar el servidor Gunicorn. Por defecto es 0.0.0.0 para que así este disponible para todos los usuarios de la red.
-o <>	Para establecer el puerto en el que publicar el servidor Gunicorn. Por defecto es 5000.
-e <>	Para establecer el fichero de inicio para el servidor Gunicorn. Por defecto tdvwebAPI.aplicacion.
-v <>	Para establecer la variable que debe de ejecuar el servidor Gunicorn. Por defecto create_app().
-t <>	Para instalar los requirements necesarios desde un fichero de requirements.
-u <>	Para lanzar el comando de ejecución para el Docker que queramos.
-n <>	Para añadir variables de entorno.

Tabla 18. Parámetros del fichero Bash

3.7.Seguimiento y control

En este apartado analizaremos el desarrollo de la iteración, así como sus desviaciones con respecto a la planificación inicial.

3.7.1. Seguimiento de cambios

Durante el desarrollo de esta iteración han surgido los siguientes cambios en el alcance de la iteración que se pueden ver en la Tabla 19, todos los cambios propuestos han sido aceptados. Estos cambios tendrán la siguiente forma, **CD – XX (Cambios dinámico)**:

Cambios			
Código	**Descripción**	**Impacto**	**Tiempo estimado**
CD-01	Desarrollo de fichero bash para lanzar el servidor.	Medio	7 h
CD-02	Desarrollo de la carga dinámica de la metainformación.	Medio	7 h
CD-03	La configuración de los modelos y la del log deben de estar en el mismo fichero de configuración.	Bajo	20 min

Tabla 19. Cambios surgidos en la iteración 3

CD -01 – Desarrollo de fichero bash para lanzar el servidor:

Durante el desarrollo de esta iteración, en el momento de lanzar el servidor hemos detectado la necesidad de desarrollar un fichero Bash para poder lanzarlo desde la imagen Docker que generemos. Este cambio corresponde con el requisito funcional RFD -10.

CD -02 – Desarrollo de la carga dinámica de la metainformación:

La empresa mientras desarrollaba la carga dinámica propuso que la carga dinámica también cargara la metainformación además de los modelos. Este cambio corresponde con el requisito funcional RFD -11.

CD -03 - La configuración de los modelos y la del log deben de estar en el mismo fichero de configuración:

La empresa consideró que solo debía haber un único fichero de configuración, tras ver que en un primer momento existían dos ficheros de configuración diferentes, uno para los modelos y otro para el log de la API. Este cambio corresponde con el requisito no funcional RND -09.

3.7.2. Seguimiento del alcance

Al comenzar la iteración se establecieron una serie de requisitos los cuales se iban a tratar de desarrollar a lo largo de la iteración, es por ello que debemos analizar la situación del alcance al finalizar la iteración.

3.7.3. Requisitos alcanzados

Se han cumplido satisfactoriamente todos los requisitos especificados en el apartado "3.2. Alcance" pese a haber sido cambiado el alcance de la iteración.

3.7.4. Entregables generados

Se han desarrollado el siguiente entregable:

- **RND-1 Documentación:** contendrá las especificaciones del servicio web desarrollado en esta iteración, así como un manual de su uso e implementación del servicio en producción.

3.8. Problemas

Durante la realización de esta componente nos encontramos con el siguiente problema, el cual supuso un retraso en la planificación:

Problemas de concurrencia con la carga dinámica en el servidor de producción (04/05/2020 – 06/05/2020):

Al probar la API con el servidor Gunicorn y emplear más de un --worker (el número de procesos de trabajo) nos dimos cuenta de que en alguno de los workers no se cargaban los modelos ni la metainformación correctamente. La solución por la que hemos optado ha sido emplear áreas de exclusión mutua mediante locks de Python en los métodos de carga para que solo esa parte al iniciar el servidor funcione de modo secuencial. Una vez cargado sí se pueda emplear concurrencia para tratar las peticiones.

4. Seguimiento y control

4.1.Seguimiento del tiempo (Cronograma real)

La siguiente tabla (Diagrama 3) muestra el diagrama de Gantt real del proyecto. En él encontramos la estimación inicial de semanas, en color granate, y la estimación de semanas reales, en color azul, para completar cada una de las tareas y su distribución en el tiempo. De esta forma podremos comprobar si han ocurrido desviaciones en el tiempo.

No tuvimos ningún retraso en el desarrollo de las dos primeras iteraciones, pero sí en el inicio de la primera iteración como ya se mencionó en el apartado Plan de contingencia (Riesgos), los cambios surgidos en la iteración tres que modificaron el alcance de esta iteración también hicieron que se retrasara su finalización, fue necesaria una reunión adicional con el tutor de la empresa con respecto a las previstas para solventar estos cambios.

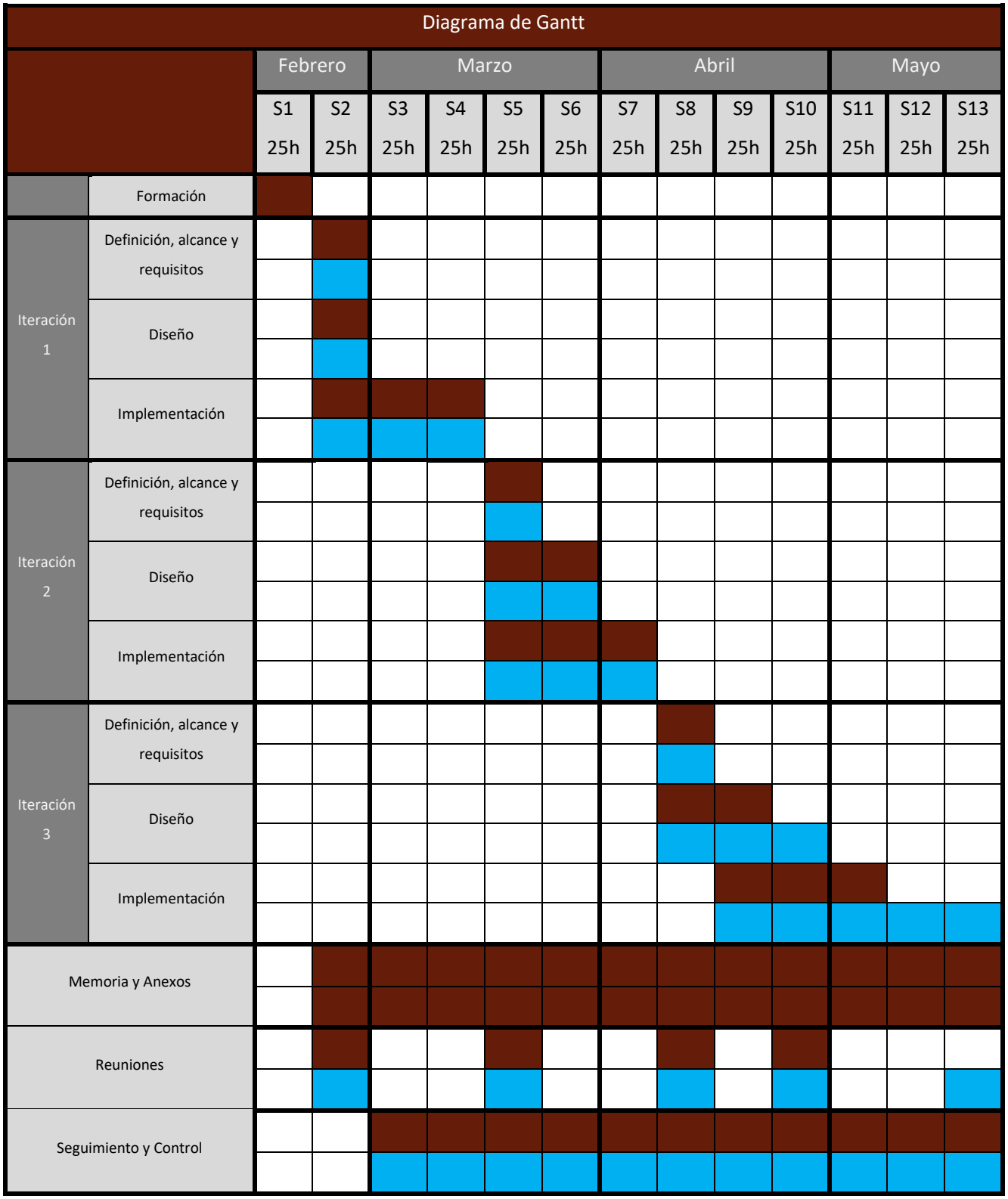

Diagrama 3. Diagrama de Gantt real

5. Conclusiones

En primer lugar, es importante destacar que el objetivo del TFG era desarrollar y publicar una API para publicar servicios y modelos de inteligencia artificial y ese objetivo lo hemos podido cumplir en los plazos previstos con alguna desviación en la planificación.

La empresa, por medio de su tutor de prácticas, también ha expresado su satisfacción con el producto desarrollado, y de hecho tiene intención de hacerlo público para uso interno de la empresa.

También me gustaría mencionar que, tras terminar el periodo de realización del TFG en la empresa, esta me ofreció un puesto de trabajo. Actualmente me encuentro trabajando en ella.

Como trabajo futuro se plantearía el añadir modelos emocionales en otros idiomas, además de una mejora en la carga dinámica para hacerla más eficiente y robusta. Así como añadir nuevos servicios a la API según fueran siendo necesarios.

También me gustaría destacar y agradecer la comunicación con mi tutor de prácticas en la empresa, que siempre me ha estado apoyando en todas las fases del desarrollo, quitándole importancia a problemas surgidos. Problemas que para mí en un principio parecían de más envergadura de la que en realidad tenían, aprendiendo gracias a este proceso a ser más optimista en cuanto a mis conocimientos adquiridos, porque pese a mi reciente incorporación al mundo laboral, tengo más conocimiento del que me presuponía inicialmente por mi falta de experiencia en este mundo. Además de ayudarme a ser más independiente animándome cuando él lo consideraba, dejándome equivocarme para que así aprendiera de mis fallos.

Agradecer también a mi tutor de la Universidad de La Rioja, Jesús María Aransay Azofra, que me ha dado un apoyo constante durante todo el tiempo que he estado en la empresa (ya que también ha sido mi tutor de prácticas), abriéndome su puerta siempre que se lo solicitaba, animándome en los momentos de más agobio.

6. Índice de recursos

Diagramas

Tablas

Imágenes

7. Bibliografía

Ley de Moore:

https://www.intel.es/content/www/es/es/silicon-innovations/moores-law-technology.html

Atlassian (Confluence, Jira):

https://www.atlassian.com/es/software/jira

Teoría de los Big-5:

https://positivepsychology.com/big-five-personality-theory/

Visual Studio:

https://visualstudio.microsoft.com/es/

Python:

https://www.python.org/

Flask:

https://flask.palletsprojects.com/en/1.1.x/

Flasgger:

https://github.com/flasgger/flasgger

TensorFlow:

https://www.tensorflow.org/

scikit-learn:

https://scikit-learn.org/stable/

Joblib:

https://joblib.readthedocs.io/en/latest/auto_examples/serialization_and_wrappers.html

Docker:

https://www.docker.com/

Kubernetes:

https://kubernetes.io/es/

Stack Overflow:

https://stackoverflow.com/

Pyformance:

https://pyformance.readthedocs.io/en/latest/

LIWC:

https://dialnet.unirioja.es/servlet/articulo?codigo=4227297

8. Anexo

Actas de reuniones

The Demanda Valley

Acta de reunión

24/02/2020
Lugar: Oficina de la empresa (Logroño)
Hora inicio: 10:00
Hora fin: 10:30

Orden del día

Análisis de la primera iteración. Definición de los requisitos.

Lista de asistentes

Jesús Navajas Briones
Millán Santamaría Sacristán

Conclusiones

Se determinaron los requisitos a realizar, así como las exclusiones, el alcance y los entregables.

The Demanda Valley

Acta de reunión

16/03/2020
Lugar: Online
Hora inicio: 10:00
Hora fin: 11:00

Orden del día

Revisión de la iteración uno. Análisis de la segunda iteración. Definición de los requisitos.

Lista de asistentes

Jesús Navajas Briones
Millán Santamaría Sacristán

Conclusiones

Se validó la primera iteración, se determinaron los requisitos a realizar, así como las exclusiones, el alcance y los entregables de la segunda iteración.

The Demanda Valley

Acta de reunión

13/04/2020
Lugar: Online
Hora inicio: 10:00
Hora fin: 11:00

Orden del día

Revisión de la segunda iteración. Análisis de la tercera iteración. Definición de los requisitos.

Lista de asistentes

Jesús Navajas Briones
Millán Santamaría Sacristán

Conclusiones

Se validó la segunda iteración, se determinaron los requisitos a realizar, así como las exclusiones, el alcance y los entregables de la tercera iteración.

The Demanda Valley

Acta de reunión

20/04/2020
Lugar: Online
Hora inicio: 10:00
Hora fin: 10:30

Orden del día

Cambio en la definición del alcance de los requisitos.

Lista de asistentes

Jesús Navajas Briones
Millán Santamaría Sacristán

Conclusiones

Se determinaron los nuevos requisitos que de debían de incluir en el alcance de la iteración tres.

The Demanda Valley

Acta de reunión

18/05/2020
Lugar: Online
Hora inicio: 10:00
Hora fin: 10:30

Orden del día

Revisión de la tercera iteración.

Lista de asistentes

Jesús Navajas Briones
Millán Santamaría Sacristán

Conclusiones

Se validó el cumplimiento de los requisitos de la tercera iteración.

Ejemplo de salida del modelo Emocional


```
{
 "LWC": {
  "SCL": {
   "SCL": 0.054093567251461985,
   "FRD": 0,
   "FML": 0,
   "HMS": 0.0014619883040935672
  },
  "AFCTP": {
   "AFCTP": 0.038011695906432746,
   "EMPOS": 0.027777777777777776,
   "EMNG": 0.008771929824561403,
   "AXT": 0.0014619883040935672,
   "ANG": 0.0014619883040935672,
   "SDS": 0
  },
  "CGPR": {
   "CGPR": 0.2236842105263158,
   "ISGT": 0.027777777777777776,
   "CSTN": 0.013157894736842105,
   "DCPC": 0.023391812865497075,
   "TTTV": 0.03070175438596491,
   "CTT": 0.01023391812865497,
   "IHBTN": 0.0043859649122807015,
   "ICSV": 0.05555555555555555,
   "EXCS": 0.011695906432748537
  },
  "PCTP": {
   "PCTP": 0.029239766081871343,
   "SEE": 0.01023391812865497,
   "HEAR": 0.01023391812865497,
   "FEEL": 0
  },
  "BIOPR": {
   "BIOPR": 0.01023391812865497,
   "BODY": 0.005847953216374269,
   "HLT": 0.0014619883040935672,
   "SEX": 0.0014619883040935672,
   "INGT": 0.0029239766081871343
  },
  "RLTV": {
   "RLTV": 0.097953216374269,
   "MTN": 0.021929824561403508,
   "SPC": 0.03654970760233918,
   "TIME": 0.04093567251461988
```

```
  },
  "WRK": 0.04093567251461988,
  "ACHV": 0.033625730994152045,
  "LSR": 0.01608187134502924,
  "HOME": 0.0014619883040935672,
  "MNY": 0.004385964912280705,
  "RLGN": 0,
  "DEATH": 0.0014619883040935672,
  "ASNT": 0,
  "NFLNC": 0,
  "FILL": 0
 },
 "EMO": {
  "B5R": {
   "OPE": 0.6038917899131775,
   "CON": 0.5657619833946228,
   "EXT": 0.5510221719741821,
   "AGR": 0.5914007425308228,
   "NEO": 0.4652288258075714
  },
  "NER": {
   "CHA": 0.7254202961921692,
   "CLO": 0.766172468662262,
   "CUR": 0.7516517043113708,
   "EXC": 0.6839134097099304,
   "HAR": 0.8050354719161987,
   "IDE": 0.5399019122123718,
   "LIB": 0.738396167755127,
   "LOV": 0.7414917349815369,
   "PRA": 0.7397951483726501,
   "SEX": 0.6364030241966248,
   "STA": 0.7152085304260254
  },
  "VAR": {
   "CON": 0.6630173325538635,
   "OPC": 0.7360683679580688,
   "HED": 0.7964732050895691,
   "SEN": 0.6322608590126038,
   "STR": 0.8333975672721863
  },
  "FCR": {
   "ADV": 0.46637365221977234,
   "ART": 0.5921856164932251,
   "EMT": 0.6364400386810303,
   "IMG": 0.7813779711723328,
   "INT": 0.5673928260803223,
   "LIB": 0.4505443572998047,
```

```
      "ACHV": 0.598503589630127,
      "CAU": 0.4286826550960541,
      "DTFL": 0.687860906124115,
      "ODRL": 0.4067998230457306,
      "DSCP": 0.6193559765815735,
      "SEF": 0.7573388814926147,
      "ALVL": 0.5715423226356506,
      "ASRT": 0.7070531249046326,
      "CHRF": 0.6450961232185364,
      "EXSK": 0.6230044960975647,
      "FND": 0.6055151224136353,
      "GGS": 0.49730461835861206,
      "ALT": 0.7100071310997009,
      "COP": 0.5379616022109985,
      "MTD": 0.3633725941181183,
      "MRT": 0.6277500987052917,
      "SYM": 0.697189211845398,
      "TRT": 0.6009612679481506,
      "ANG": 0.5728490948677063,
      "ANX": 0.6695689558982849,
      "DPS": 0.46638917922973633,
      "IMDT": 0.47534051537513733,
      "SCNS": 0.5186339616775513,
      "VLB": 0.5596943497657776
    }
  }
}
```

Ejemplo de la salida de las métricas


```
{
 "CONSULTAMETADATA": {
  "15m_rate": 0.00028776486793499507,
  "1m_rate": 0.0002877648657237664,
  "50_percentile": 0.00035285949707031250,
  "5m_rate": 0.00028776486714527055,
  "75_percentile": 0.00035285949707031250,
  "95_percentile": 0.00035285949707031250,
  "999_percentile": 0.00035285949707031250,
  "99_percentile": 0.00035285949707031250,
  "avg": 0.00035285949707031250,
  "count": 1,
  "max": 0.00035285949707031250,
  "mean_rate": 0.0002877648652499317,
  "min": 0.00035285949707031250,
  "std_dev": 0,
  "sum": 0.00035285949707031250
 },
 "CONSULTAMETADATACOUNTER": {
  "15m_rate": 0.0002877648745686813,
  "1m_rate": 0.00028776487291025973,
  "5m_rate": 0.0002877648737197274,
  "count": 1,
  "mean_rate": 0.0002877648703436549
 },
 "CONSULTAMETRICA": {
  "15m_rate": 0,
  "1m_rate": 0,
  "50_percentile": 0,
  "5m_rate": 0,
  "75_percentile": 0,
  "95_percentile": 0,
  "999_percentile": 0,
  "99_percentile": 0,
  "avg": 0,
  "count": 0,
  "max": -2147483647,
  "mean_rate": 0,
  "min": 2147483647,
  "std_dev": 0,
  "sum": 0
 },
 "CONSULTAMETRICACOUNTER": {
  "count": 1
 },
 "CONSULTAMETRICAMETER": {
  "15m_rate": 0,
```

"1m_rate": 0,
"5m_rate": 0,
"count": 1,
"mean_rate": 8289.138339920948
}
}

Fichero yalm para Kubernetes

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: tdvwebapi
  labels:
    app: tdvwebapi
spec:
  replicas: 1
  selector:
    matchLabels:
      app: tdvwebapi
  template:
    metadata:
      labels:
        app: tdvwebapi
    spec:
      initContainers:
      # This container clones the desired git repo to the EmptyDir volume.
        - name: git-clone
          image: alpine/git # Any image with git will do
          args: ["clone","--single-branch","--branch","develop","--depth","1","<<URL para clonar el
repositorio con el tocken de acceso>>","/repo"]
          volumeMounts:
            - name: src
              mountPath: /repo
      containers:
        - name: flaskgunicorn
          image: "tdvregistry.azurecr.io/webtdv:latest"
          imagePullPolicy: IfNotPresent
          env:
          - name: TDV_CONFIG
            value: /src/run/etc/webservS/config.json
          args: ["-p", "/src/source/lib","-w","4","-i","0.0.0.0","-o","5000","-e","webservS.aplicacion","-
v","create_app()"]
          ports:
          - name: http
            containerPort: 5000
            protocol: TCP
          volumeMounts:
          - name: src
            mountPath: /src
          resources:
            {}
          livenessProbe:
            null
          readinessProbe:
            null
      volumes:
        - name: src
          emptyDir: {}
```

```
---
apiVersion: v1
kind: Service
metadata:
  name: tdvwebapi
spec:
  type: ClusterIP
  ports:
    - port: 80
      targetPort: 5000
      protocol: TCP
      name: http
  selector:
    app: tdvwebapi
```